

COMPUTE-IN-MEMORY PRIMITIVES FOR ENERGY-EFFICIENT MACHINE LEARNING

by

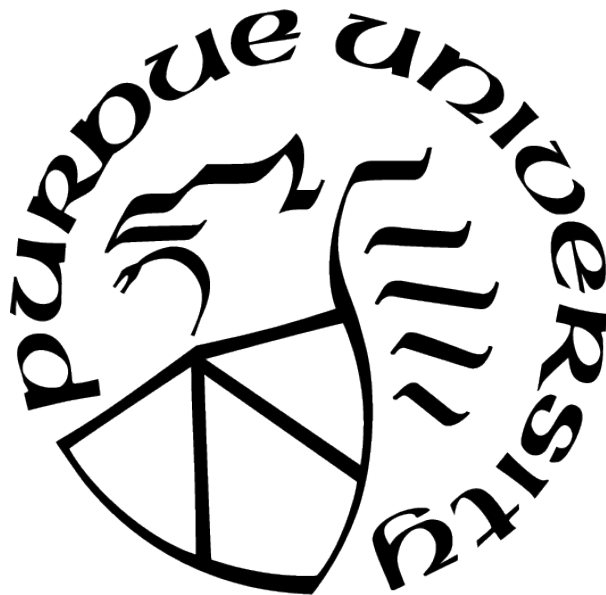
Amogh Agrawal

A Dissertation

Submitted to the Faculty of Purdue University

In Partial Fulfillment of the Requirements for the degree of

Doctor of Philosophy



School of Electrical and Computer Engineering

West Lafayette, Indiana

August 2021

**THE PURDUE UNIVERSITY GRADUATE SCHOOL
STATEMENT OF COMMITTEE APPROVAL**

Dr. Kaushik Roy, Chair

School of Electrical and Computer Engineering

Dr. Anand Raghunathan

School of Electrical and Computer Engineering

Dr. Sumeet K. Gupta

School of Electrical and Computer Engineering

Dr. Vijay Raghunathan

School of Electrical and Computer Engineering

Approved by:

Dr. Dimitrios Peroulis

This work is dedicated to Baba, Amma, Mausi, Nanaji and Nani.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Prof. Kaushik Roy for his continued support and guidance during my PhD study and research. His exceptional mentorship helped me shape myself as a researcher and allowed me to explore various interesting topics, while building my skills and critical thinking. I could not have imagined a better mentor for my PhD study.

I would also like to thank my committee members Prof. Anand Raghunathan, Prof. Sumeet Gupta, and Prof. Vijay Raghunathan for their encouragement and always being easily approachable. Their invaluable advice at each step of my PhD journey helped me stay motivated and focused.

My sincere thanks also goes to Dr. Ajey Jacob, Dr. Steven Woo and Dr. Thomas Vogelsang for offering me the summer internship opportunities for working on very exciting projects, which helped me broaden my scope and build network outside academia.

I thank all my fellow NRL members and alumni. Special thanks to Dr. Akhilesh Jaiswal for helping me get started on my first project, Dr. Ankit Sharma for all his help initially, and Dr. Minsuk Koo for the very helpful insights during the tapeout process. Also I thank all my collaborators: Mustafa, Nitin, Dr. Indranil Chakraborty, Dr. Deboleena Roy, Dr. Aayush Ankit, Dr. Chankyu Lee, Adarsh, Sangamesh, Dong Eun, Tanvi, Deepika, Utkarsh, Shubham, Eunseon, and Dr. Cheng Wang, with whom I had the pleasure of working, having stimulating discussions, and all the fun times together.

I would also like to acknowledge Nicole for efficiently administering C-BRIC and always being ready to offer help with almost anything.

Lastly, but most importantly, I would like to thank my parents Dr. Vivek Agrawal and Dr. Jolly, my sister Dr. Akanksha Agrawal, my brother-in-law Dr. Shrihari Kulkarni, and all friends and family for their unconditional love and support.

TABLE OF CONTENTS

LIST OF TABLES	11
LIST OF FIGURES	12
ABSTRACT	21
1 INTRODUCTION	23
2 ENABLING IN-MEMORY BOOLEAN COMPUTATIONS IN STANDARD 8T SRAM ARRAYS	27
2.1 Introduction	27
2.2 In-Memory Computations in 8-Transistor SRAM Bit-Cells	31
2.2.1 8-Transistor SRAM: NOR operation	33
2.2.2 8-Transistor SRAM: NAND operation	35
2.2.3 8 Transistor SRAM: Voltage Divider Scheme for IMP and XOR gates	36
2.2.4 Proposed ‘read-compute-store’ (RCS) scheme	40
2.3 8 ⁺ Transistor Differential Read SRAM	42
2.4 Discussions	45
2.5 X-SRAM based non-standard von-Neumann Computing for AES Encryption	47
2.5.1 Simulation Methodology	48
2.5.2 Results and Discussion	49
2.6 Conclusion	50

3	ACCELERATING BINARY CONVOLUTIONAL NEURAL NETWORKS IN 10T SRAM ARRAYS	51
3.1	Introduction	51
3.2	In-memory Binary Convolution – Proposal-A	54
3.2.1	Circuit Description	55
3.2.2	Dual Read-Wordline based Dual-stage ADC	57
3.2.3	Sectioned Memory Array for Parallel Computing	61
3.2.4	Results	63
3.3	In-memory Binary Convolution – Proposal-B	64
3.3.1	Bitwise XNORs	65
3.3.2	Popcount	66
3.3.3	Results	67
3.4	System-level Evaluation Framework for BNN	68
3.4.1	Simulation Methodology	69
3.4.2	Mapping Weights and Activations to Xcel-RAM	70
3.4.3	Results and Discussion	72
3.5	Conclusion	74
4	ENABLING DOT-PRODUCT COMPUTATIONS IN STANDARD 8T-SRAM ARRAYS USING CHARGE ACCUMULATION AND SHARING	75
4.1	Introduction	75
4.2	Related Works	78

4.3	Charge Sharing based In-Memory Dot-Product Operation	79
4.3.1	8T-SRAM: Structure and Operation	80
4.3.2	8T-SRAM: Charge sharing based Dot-Product Operation	81
4.3.3	SPICE Characterization	82
4.3.4	Self-Compensation	84
4.3.5	Compensating for Transistor Non-linearity	85
4.4	System Integration of CASH-RAM for Accelerating Ternary Weight Neural Networks	87
4.4.1	Cache Integration	88
4.4.2	Subarray Details	88
4.4.3	Data Mapping	89
4.5	Results	89
4.5.1	Experimental methodology	90
4.5.2	Impact of Non-idealities on Classification Accuracy	90
4.5.3	Energy, Delay and Area Estimates	92
4.6	Conclusion	97
5	LOOKUP TABLE BASED COMPUTING USING ROM-EMBEDDED SRAM . .	98
5.1	Introduction	98
5.2	RECache: Design and Operation	100
5.2.1	8T-SRAM	100

5.2.2	8 ⁺ T Differential Read SRAM	103
5.3	Evaluating RECache on realistic workloads	105
5.4	Conclusions	107
6	SPIKING NEURAL NETWORK ACCELERATION USING LOOKUP TABLE BASED IN-MEMORY-COMPUTING	108
6.1	Introduction	108
6.2	Background	110
6.2.1	ROM-Embedded RAMs	110
6.2.2	SNN: Spiking Neural Networks	115
6.2.3	LUT based storage in R-SRAMs and R-MRAMs	116
6.3	SPARE: SNN Accelerator using ROM-embedded RAMs	117
6.3.1	SPARE Organization	117
6.3.2	Inter-layer pipelining	120
6.3.3	Processing Element (PE)	121
6.3.4	Modeling complex neuro-synaptic functionality	123
6.4	Experimental Methodology	124
6.5	Results	126
6.5.1	Energy	126
6.5.2	Area	130
6.5.3	Performance	130

6.5.4	Complex neuro-synaptic models	130
6.6	Conclusion	131
7	A 65-NM DIGITAL COMPUTE-IN-MEMORY MACRO ENABLING SPIKE-BASED SEQUENTIAL LEARNING IN 10T SRAM ARRAY	133
7.1	Introduction	133
7.2	IMPULSE: Structure and Operation	134
7.2.1	Reconfigurable Column Peripherals	136
7.2.2	In-Memory SNN Instructions	137
7.2.3	Multiple Neuron Functionalities	139
7.3	Implementation Results	139
7.4	Multi-macro Architecture	143
7.4.1	Introduction	143
7.4.2	Zero-skipping	145
7.4.3	Macro Pipeline	146
7.4.4	Support for Multiple Bit-precision	148
7.4.5	Configurations for Low and High Fan-in CNN Layers	149
7.4.6	Timestep Pipelining: Leveraging Additional Weight Re-use	151
7.4.7	Preliminary Results	152
7.5	Conclusion	153
8	SUMMARY AND FUTURE DIRECTIONS	155

A	CHALLENGES WITH 6T SRAM FOR ENABLING COMPUTE-IN-MEMORY .	156
A.1	Operation of 6T SRAM	156
A.2	Read Stability Challenges due to CIM	156
A.2.1	Short-circuit paths	156
A.2.2	Pseudo-write	158
	REFERENCES	159
	VITA	174
	PUBLICATIONS	175

LIST OF TABLES

2.1	Summary of proposals described in the manuscript. The table shows average energy consumption per-bit and latency for the in-memory operations on various bit-cells. Pros and cons of each proposal are also listed.	44
3.1	Benchmark Binary Neural Network [6] used for classifying CIFAR10 and SVHN datasets.	71
4.1	Hardware Parameters Description	94
4.2	Network Parameters Description	95
4.3	Energy, Delay, and Area Comparison	95
4.4	Area breakdown	96
5.1	Benchmarks used to evaluate RECache [49], [118]	105
5.2	ROM and RAM energy per-access for various array sizes obtained from CACTI.	106
7.1	Energy Efficiency of SNN over LSTM.	142
7.2	Comparison with prior works.	143

LIST OF FIGURES

1.1	(a) Intel Xeon CPU [18]. (b) NVIDIA Turing GPU [19]. (c) Google TPU [20]. (d) Eyeriss chip [21].	25
2.1	Illustration of the von-Neumann bottleneck. Frequent to-and-fro data transfers between the processor and memory units incur large energy consumption and limits the throughput. Computing within the memory array enhances the memory functionality thereby reducing the number of unnecessary transfers of data for certain class of operations like vector bit-wise Boolean logic <i>etc.</i> .	28
2.2	A summary of <i>In-Memory</i> computing schemes proposed. With respect to the 8T cell, we present bit-wise NAND, NOR and XOR operations using skewed inverter sensing. Further, we present the voltage-divider based operation of 8T-cells for IMP and XOR gates. With respect to the 8 ⁺ T-cells, we present bit-wise NAND, NOR and XOR operations using asymmetric differential SAs. Moreover, a ‘ <i>read-compute-store</i> ’ operation has been presented for both types of bit-cells.	29
2.3	a) Schematic of a standard 8T-SRAM bit-cell. In addition to the standard 6T cell, two additional transistors form the read path using a separate read bit-line (RBL). b) Single ended sensing of NAND/NOR using gated skewed inverters. Figure also shows the truth table for NAND/NOR/XOR operations. c) Timing diagram for reading NOR output of Cell 1 and Cell 2. d) Timing diagram for reading NAND output of Cell 1 and Cell 2.	32
2.4	Monte-Carlo simulations in SPICE for NAND and NOR outputs for all possible input cases – ‘00,01,10,11’, in presence of 30mV sigma variations in threshold voltage.	33
2.5	Monte-Carlo simulations across process corners (TT corner and SS corner shown) under voltage and temperature variations for NAND outputs for the borderline cases – ‘01/10’ and ‘11’. The distribution of RBL voltage is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD}	34
2.6	a) Circuit schematic of the 8T-SRAM for implementing the voltage-divider scheme. b) Equivalent circuit traced by transistors $M1 - M4$ while data is read from Cell 1 and Cell 2. c) Monte-Carlo simulations in SPICE for all possible input cases, showing the output of the two asymmetric inverters. . .	37

2.7	Monte-Carlo simulations with variations in supply voltage and temperature across process corners for the voltage-divider scheme for the case (1,1). V_{error} is defined as the difference between the RBL voltage (when both the operands are ‘1’) and the initial pre-charge voltage V_{pre} . The distribution of V_{error} is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD} . The variations in V_{pre} are also accounted for.	38
2.8	a) Proposed ‘read-compute-store’ (RCS) scheme. RWL1 and RWL2 are enabled, corresponding to the data to be computed. The computation output is selectively passed to the write-driver of that column, while simultaneously enabling the WWL3, where data is to be stored. b) Block diagram showing the RCS blocks in the memory array. The NAND of row 1 and row 2 is to be stored in row 3. c) Monte-Carlo simulations in SPICE, showing the final state of Cell 3 stores the desired output.	40
2.9	a) Circuit schematic of an 8 ⁺ T Differential SRAM bit-cell [45]. b) Timing diagram used for in-memory computations on the 8 ⁺ T Differential SRAM. c) Circuit schematic of the proposed asymmetric differential sense amplifier. . .	41
2.10	Monte-Carlo simulations in SPICE for SA outputs for all possible input cases – ‘00,01,10,11’, in presence of 30mV sigma variations in threshold voltage. .	42
2.11	Monte-Carlo simulations across process corners under V_T and temperature and supply-voltage variations for the 8 ⁺ T SRAM configuration for the cases ‘01/10’ and ‘11/00’. V_{diff} is defined as the absolute difference between the RBL and RBLB voltages at the instant when the sense amplifier is enabled. The distribution of V_{diff} is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD} . .	43
2.12	a) Thin cell layout for the standard 8T-SRAM bit-cell shown in Fig. 2.3(a). b) Thin cell layout for the 8 ⁺ T Differential SRAM bit-cell [45] illustrated in Fig. 2.9(a). Left- and right-most diffusion tracks are shared with adjacent bit-cells. The ninth transistor in Fig. 2.9(a) is common for the row and is connected at the periphery to the node ‘VX’.	46
2.13	(a) System-level implementation of a typical von-Neumann architecture with X-SRAM as the memory block. The processor, data-memory and the instruction-memory blocks are connected via a shared system bus. (b) Illustration of custom in-memory instructions added to the instruction set of the Nios-II processor. Substituting in-memory instructions reduces unnecessary read-writes into the memory.	47
2.14	Normalized number of memory accesses for various AES encryption and decryption modes and two different key-sizes, with and without using X-SRAM custom in-memory instructions. The total memory transactions are split into memory read instructions, memory write instructions and custom in-memory instructions.	48

2.15	(a) Realistic scenario for a typical system with multiple masters over a shared bus. An arbiter keeps track of the memory traffic and controls which master has access to the bus at a given point in time. (b) Data-parallelism in memory arrays. X-SRAM performs bit-wise operations throughout the row, where each row may store multiple data words. Thus multiple computations occur in parallel.	49
3.1	The 10 transistor SRAM cell featuring a differential decoupled read-port comprising of transistors M1-M2 and M1'-M2'. The write port is constituted by write access transistors connected to WWL.	54
3.2	Illustration of the binary convolution operation within the 10T-SRAM array. a) Step 1: Pseudo-read. RBLs/RBLBs are pre-charged and RWL for a row storing the input activation (A1) is enabled. Depending on the data A1, RBLs/RBLBs either discharge or stay pre-charged. The SAs are not enabled, in contrast to a usual memory read. Thus, the charge on RBLs/RBLBs represent the data A1. b) Step 2: XNOR on SL. Once the charges on RBLs/RBLBs have settled, RWL for the row storing the kernel (K1) is enabled. Charge sharing occurs between the RBLs/RBLBs and the SL, depending on the data K1. The RBLs either deposit charge on the SL, or take away charge from SL. c) The truth table for Step 2 is shown. The pull-up and pull-down of the SL follow the XNOR truth table. Moreover, since the SL is common along the row, the pull-ups and pull-downs are cumulative. Thus, the final voltage on SL represents the $XNOR + popcount$ of A1 and K1.	56
3.3	a) Dual RWL technique. b) Dual-stage ADC scheme.	57
3.4	The plot shows the final SL voltage with and without the Dual RWL approach. A larger sense margin is obtained with our Dual RWL approach, thus relaxing the constraints on the low-overhead ADC. Note that with Dual RWL technique we restrict the distinct voltage levels on SL to 32 at a time, instead of 64. However, the voltage swing on SL remains the same, thereby increasing the sense margin between the states.	58
3.5	The figure shows the timing diagrams for the dual-stage ADC scheme. The figure plots the SL voltage for various <i>popcount</i> cases. In the first-stage, the sub-class SC1-4 is determined using multiple references (0.25V, 0.5V and 0.75V). In the second-stage, charge is pumped-in/out of SL successively, depending on the SC. The number of cycles it takes for SL to reach V_{REF} are counted. V_{REF} for SC1-4 is 0.25V, 0.5V, 0.5V and 0.75V, respectively.	59

3.6	a) Typical SRAM memory array with row and column peripherals storing the activations A1-Am, and kernels K1-Kn. b) Proposed sectioned-SRAM array. By introducing switches along the RBLs, the array is divided into sections. The kernels are mapped into the sectioned-SRAM with each section storing different kernel. Once the activations are read onto the RBLs, the switches are opened, and the memory array is divided into sections. c) Since the RBLs for each section have been decoupled, one RWL in each section can be simultaneously enabled such that each section performs the binary convolution concurrently. The Row-MUX connects the corresponding SL to the sensing circuit. For example, if A1 was read onto the RBLs before sectioning, enabling the rows K1 and K2 in Section 1 and 2 respectively, we obtain A1*K1 and A1*K2 in parallel on the SLs, which are sensed by the ADC.	61
3.7	Monte-Carlo simulations. The figure plots the histogram of the second-stage output of the ADC, for various <i>popcount</i> cases, in presence of process variations. Inset: Each histogram is fitted with a Gaussian distribution. The average standard deviation of the counts is ~ 0.4359 . The trend repeats for higher <i>popcount</i> cases with modulo-8, since only the lower 3bits of the output are generated in the second-stage.	64
3.8	(a) A 10T-SRAM bitcell schematic is repeated here for convenience. (b) Timing diagram used for in-memory computing with 10T-SRAM bitcells. (c) Circuit schematic of the asymmetric differential sense amplifier. [61]	65
3.9	Modified peripheral circuitry of the SRAM array to enable binary convolution operations. It consists of two asymmetric SAs - SA_{NOR} and SA_{NAND} which pass the XNORed data vector to a bit-tree adder. The adder has $\log(N)$ layers, where N is the number of inputs to the adder. It sums the input bits to generate the <i>popcount</i>	67
3.10	(a) Modified von-Neumann architecture based on Xcel-RAM memory banks and enhanced instruction set architecture (ISA) of the processor. (b) Snippet of assembly code for performing a binary convolution operation using conventional instructions and custom instructions.	68
3.11	Mapping of weights and activations of a convolutional neural network to Xcel-RAM. The kernels and the input feature maps are flattened and stored into multiple rows in the memory array. Xcel-RAM banks have dedicated rows for storing kernels and activations.	70
3.12	Layer-wise energy consumption and latency, for running the CIFAR-10 (a-b), and SVHN (c-d) image classification benchmarks on the proposed designs, and the baseline.	72
3.13	Energy, latency and accuracy tradeoff for classifying CIFAR-10 and SVHN dataset with BNN, using the proposed techniques.	73

4.1	(a) Schematic of the standard 8T-SRAM bitcell (b) Parasitic capacitance C_{BL} and C_{SL} for an array of 8T-SRAM cells. The dotted arrows show the charge-sharing path used in our approach.	80
4.2	Data obtained from SPICE simulations. (a) Final SL voltage as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) Final SL voltage as a function of K . Three cases for $v_i=0.2,0.4,0.6V$ are shown. (c) Representation of V_{SL} as a function of ideal dot product A . The degree of spread represents the non-idealities, as illustrated in the figure taking an example of $V_{SL}=0.25V$	83
4.3	Data obtained from SPICE simulations and \hat{A} estimated from Eq. 4.8. (a) \hat{A} as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) \hat{A} as a function of K . Three cases for $v_i=0.2,0.4,0.6V$ are shown. (c) Representation of \hat{A} as a function of ideal dot product A . The degree of spread is significantly lower than in Fig. 4.2(c).	85
4.4	Data obtained from SPICE simulations and \hat{A} estimated from Eq. 4.9. (a) \hat{A} as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) \hat{A} as a function of K . Three cases for $v_i=0.2,0.4,0.6V$ are shown. (c) Representation of \hat{A} as a function of ideal dot product A . The degree of spread has further reduced significantly than in Fig. 4.3(c).	86
4.5	CASH-RAM system integration for accelerating TWNNs. A typical multi-bank cache hierarchy is shown on the right, where each bank consists of multiple sub-arrays. The subarray is shown on the left, with additional peripheral circuitry to augment dot-product computations within the cache.	87
4.6	Classification accuracy obtained on MNIST and CIFAR-10 datasets for different benchmarks, for with and without self-compensation. Approach-1 corresponds to self-compensation while Approach-2 corresponds to compensating for transistor non-linearity.	91
5.1	(a) Schematic of a standard 8T-SRAM bitcell. Transistors M1 and M2 form the decoupled read port of the SRAM cell. (b) Proposal-A for RECache. The cell has an extra RWL, and the ROM data stored is ‘1’ if the access transistor is connected to RWL1, and ‘0’ if it is connected to RWL2. The node voltages for RAM and ROM modes of operation are listed. (c) Proposal-B for RECache. This configuration has two SLs, instead of two RWLs. ROM data stored is ‘1’ if the access transistor is connected to SL1, and ‘0’ if it is connected to SL2.	99
5.2	Timing diagrams generated from HSPICE for Proposal-A,B. The correct ROM Output data is generated, without disturbing the RAM data stored in the cell. The circuit shown in Fig. 5.1(b-c) are simulated.	100

5.3	(a) Thin cell layout of a standard 8T-SRAM cell. The circled contacts, RWL and SL, are common to adjacent bitcells in the horizontal and vertical direction, respectively. (b) Thin cell layout of the 8 ⁺ T-SRAM bit-cell. The circled contacts (VX) on either side of the bitcell are shared by adjacent bitcells.	101
5.4	(a) Schematic of the differential 8 ⁺ T-SRAM bitcell. Transistors M1 and M2 form a differential read port decoupled from the 6T write port. The ninth transistor connected to RWL is common for the entire row. (b) RECache using the differential 8 ⁺ T-SRAM. The connection of M1 to either VX1 or VX2 determines the ROM bit stored in the cell. The ROM retrieval process is exactly same as the 8T-RECache. The node voltages for ROM and RAM mode of operation are listed.	102
5.5	Timing diagrams obtained from HSPICE. The correct ROM Output data is generated, without disturbing the RAM data stored in the cell. The circuit shown in Fig. 5.4(b) was simulated.	103
5.6	Normalized cache miss-rate for various benchmarks – artificial neural network (ANN), spiking neural network (SNN) and advanced encryption standard (AES), with RECache and standard SRAM caches.	104
6.1	R-SRAM Schematic: Standard 6T-SRAM embedded with ROM. The only difference is the addition of extra word-line (WL1 and WL2) to embed ROM functionality.	109
6.2	Operation of R-SRAM in a) Normal RAM Mode and b) ROM Mode.	111
6.3	R-MRAM Schematic: Standard STT-MRAM array with two bit-lines (BL1 and BL0) to embed ROM functionality. The peripheral circuitry for RAM and ROM mode of operation is highlighted.	112
6.4	Typical SNN dynamics. The input spikes are modulated by the synaptic weights, and the accumulated synaptic current is fed to the neuron. The neuron integrates the current and outputs a spike (fires) once its membrane potential exceeds a threshold.	113
6.5	Storage of LUTs for various functions within the same ROM-Embedded RAM array. The starting address for each type of LUT is predefined. An offset address (calculated from the input) is added to the starting address to perform the table lookup from the R-SRAM/R-MRAM. The number of memory rows required by each LUT type is predefined based on the desired precision of the transcendental function to be stored.	113

6.6	Block level diagram of SPARE. (a) Figure shows how a deep neural network is mapped to a 2-D array of PEs connected together. The global memory stores the spiking events at every layer output, and broadcasts them to the input of next layer. (b) Figure zooms into the logical diagram of the PE. It consists of a ROM-embedded RAM to store the state variables along with LUTs of synapse, neuron and synaptic plasticity models, computation core to generate output spikes, input buffers to store incoming spike broadcast, event controller to schedule memory transactions, state updater to update the entries in the memory, and an output buffer to store the output spikes generated.	114
6.7	Mapping of CNNs in SPARE: The input map is window-split based on the kernel size of the particular layer. These are then broadcast to all PEs mapped to that layer. Each PE stores different kernels, and process the data in parallel as they receive the inputs in a window split-manner. Each PE computes part of the output feature map, highlighted through color coding of PEs in figure. The output is rearranged and stored back to the global memory unit.	118
6.8	Logical flow diagram of the event controller, describing SNN computations performed in the PE. The subsequent computation is subdivided into three main blocks. 1) Synapse model block: computes output synaptic current. 2) Neuron model block: keeps track of the membrane potential of output neurons. 3) Plasticity model block: updates synaptic weights during the training phase. This block is skipped during the inference phase.	119
6.9	Timing diagram illustrating the inter-layer pipelining in SPARE. As soon as the PEs receive and buffer the input data, they start processing. Meanwhile, data for PEs mapped to subsequent layers is transmitted. Since the data transfer time is small compared to the computation time within the PE, all PEs process data in parallel.	120
6.10	Differential equations describing the dynamics of neurons and an LUT based approach to implement them in SPARE. (a) Leaky-integrate-fire neuron (b) Izhikevich neuron (c) Hodgkin-Huxley neuron	122
6.11	Energy and latency for read-write accesses from all designs considered – SRAM, R-SRAM, STT-MRAM, and R-MRAM. (* ROM Read for R-SRAM includes additional overhead of buffering RAM, retrieving ROM data and storing back the buffered RAM data, as described in Section 6.2.1).	123
6.12	uArchitecture design parameters used for simulations.	123
6.13	SNN benchmarks used in SPARE evaluation [118], [129]. The figure tabulates the number of PEs required, memory requirement, and the RAM/ROM content for each benchmark and neuron model.	124

6.14	Normalized energy consumption for a) Training phase and b) Inference phase, for benchmarks ‘MNIST1-3’. The simulations are performed for max firing rate $fp = 0.4$ and 1. The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.	126
6.15	Normalized energy consumption for benchmarks ‘MNIST4’ and ‘CIFAR10’. The simulations are performed for max firing rate $fp = 0.4$ and 1. The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.	127
6.16	per-PE area with SRAM, R-SRAM, STT-MRAM and R-MRAM as memory units (for iso-storage).	129
6.17	Normalized energy consumption for using Hodgkin-Huxley neuron models on SNN benchmarks. The simulations are performed for max firing rate $fp = 0.4$. The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.	129
7.1	(a) LSTM having hidden state (h^t , C^t) for processing sequential tasks. (b) Intrinsic temporal dynamics of neuron membrane potentials (V_{MEM}^t) in SNNs for processing sequential tasks.	134
7.2	Limitations of current digital SNN hardware accelerators and our proposed approach of fused weight and membrane potential CIM SRAM.	134
7.3	(a) Organization of the fused W_{MEM}/V_{MEM} 10T-SRAM macro. (b) Mapping of FC and Conv layers on the proposed macro.	135
7.4	Detailed description of the reconfigurable column peripherals.	136
7.5	Illustration of supported in-memory SNN instructions.	137
7.6	Multiple neurons can be implemented using in-memory instructions.	138
7.7	Die micrograph, area breakdown, and Shmoo plot for CIM operations. . . .	139
7.8	(a) Measured average power and energy-efficiency for AccW2V instruction. (b) SNN architecture and accuracy on IMDB and MNIST datasets.	140
7.9	Progression of the final output neuron’s membrane potential with timesteps, where each word is presented to the SNN for 10 timesteps.	141
7.10	(a) Average sparsity obtained at each layer of SNN for each timestep. (b) Measured EDP per-neuron per-timestep with varying sparsity.	142

7.11	Limited fan-in of single macro (left). Multi-macro architecture is shown on the right consisting of compute and neuron macros and moving partial V_{MEM} among them to compute the final spikes (right). The block diagram showing each processing element (bottom).	144
7.12	Zero-skipping using leading-one detector.	145
7.13	Macro Pipeline. Pipeline stages Read, Compute and Store corresponding to the column peripheral blocks SINV, BLFA and CWD, respectively (right). Timing diagram of the macro processing is shown on the left.	147
7.14	Amortizing the reconfigurability overhead for processing odd and even columns, by maintaining two queues having appropriate depth. The graph on the right plots the energy per operation as a function of number of consecutive odd/even operations.	148
7.15	Variable bit-precisions can be supported by introducing RBL switches to decouple the RBL between the W_{MEM} and V_{MEM} subarrays.	148
7.16	Two supported configurations to efficiently run low fan-in and high fan-in CNN layers with high throughput.	149
7.17	Timing diagram showing the top-level timestep pipelining among the macros (top). Once all timesteps are performed for one group of neurons, we can start processing the next group and so on, thereby leveraging weight re-use across neurons and across timesteps. The plot (bottom right) shows the dependency of compute cycles on the data-sparsity, showing the benefits of zero-skipping.	150
7.18	Energy-efficiency of the proposed multi-macro architecture, as a function of data-sparsity and the bit-precision, for the two proposed configurations.	151
7.19	Throughput of the proposed multi-macro architecture, as a function of data-sparsity and the bit-precision, for the two proposed configurations.	152
7.20	Energy breakdown for two levels of sparsity (75% and 99%).	152
A.1	Schematic showing a 6T-SRAM array. The red and blue arrows show the current path and the charge discharge path, respectively, causing read-stability issues for performing in-memory computing.	157
A.2	Staggered activation of wordlines to avoid short circuit paths, leading to bit-line discharge [90]	158

ABSTRACT

Machine Learning (ML) workloads, being memory and compute-intensive, consume large amounts of power running on conventional computing systems, restricting their implementations to large-scale data centers. Thus, there is a need for building domain-specific hardware primitives for energy-efficient ML processing at the edge. One such approach is in-memory computing, which eliminates frequent and unnecessary data-transfers between the memory and the compute units, by directly computing the data where it is stored. Most of the chip area is consumed by on-chip SRAMs in both conventional von-Neumann systems (e.g. CPU/GPU) as well as application-specific ICs (e.g. TPU). Thus, we propose various circuit techniques to enable a range of computations such as bitwise Boolean and arithmetic computations, binary convolution operations, non-Boolean dot-product operations, lookup-table based computations, and spiking neural network implementation – all within standard SRAM memory arrays.

First, we propose X-SRAM, where, by using skewed sense amplifiers, bitwise Boolean operations such as NAND/NOR/XOR/IMP etc. can be enabled within 6T and 8T SRAM arrays. Moreover, exploiting the decoupled read/write ports in 8T SRAMs, we propose read-compute-store scheme where the computed data can directly be written back in the array simultaneously.

Second, we propose Xcel-RAM, where we show how binary convolutions can be enabled in 10T SRAM arrays for accelerating binary neural networks. We present charge sharing approach for performing XNOR operations followed by a population count (popcount) using both analog and digital techniques, highlighting the accuracy-energy tradeoff.

Third, we take this concept further and propose CASH-RAM, to accelerate non-Boolean operations, such as dot-products within standard 8T-SRAM arrays by utilizing the parasitic capacitances of bitlines and sourcelines. We analyze the non-idealities that arise due to analog computations and propose a self-compensation technique which reduces the effects of non-idealities, thereby reducing the errors.

Fourth, we propose ROM-embedded caches, RECache, using standard 8T SRAMs, useful for lookup-table (LUT) based computations. We show that just by adding an extra word-line

(WL) or a source-line (SL), the same bit-cell can store a ROM bit, as well as the usual RAM bit, while maintaining the performance and area-efficiency, thereby doubling the memory density. Further we propose SPARE, an in-memory, distributed processing architecture built on RECache, for accelerating spiking neural networks (SNNs), which often require high-order polynomials and transcendental functions for solving complex neuro-synaptic models.

Finally, we propose IMPULSE, a 10T-SRAM compute-in-memory (CIM) macro, specifically designed for state-of-the-art SNN inference. The inherent dynamics of the neuron membrane potential in SNNs allows processing of sequential learning tasks, avoiding the complexity of recurrent neural networks. The highly-sparse spike-based computations in such spatio-temporal data can be leveraged for energy-efficiency. However, the membrane potential incurs additional memory access bottlenecks in current SNN hardware. IMPULSE tries to tackle the above challenges. It consists of a fused weight (W_{MEM}) and membrane potential (V_{MEM}) memory and inherently exploits sparsity in input spikes. We propose staggered data mapping and re-configurable peripherals for handling different bit-precision requirements of W_{MEM} and V_{MEM} , while supporting multiple neuron functionalities. The proposed macro was fabricated in 65nm CMOS technology. We demonstrate a sentiment classification task from the IMDB dataset of movie reviews and show that the SNN achieves competitive accuracy with only a fraction of trainable parameters and effective operations compared to an LSTM network.

These circuit explorations to embed computations in standard memory structures shows that on-chip SRAMs can do much more than just store data and can be re-purposed as on-demand accelerators for a variety of applications.

1. INTRODUCTION

In the past decade, we have seen a tremendous growth in Machine Learning (ML) algorithms, especially Deep Neural Networks (DNNs). DNNs have been shown to be extremely effective for various cognitive applications, such as classification, recognition, detection and autonomous systems, which are being adopted into various disciplines [1], [2]. The primary reason for their exponential growth and widespread adoption in the past decade can be attributed to the advancements in computational power and resources [3]. Availability of powerful large-scale CPU and GPU servers and clusters enabled execution of computationally expensive DNN models, leading to superior performance [3]. Even today, the size of the state-of-the-art DNNs grows exponentially [4]. Although large-scale data-centers having multiple CPU clusters and GPUs, enable large parallelism and faster execution of DNNs, they are extremely power hungry. This is because DNN compute models are inherently different from general-purpose workloads and are immensely memory- and compute-intensive. Running data-intensive applications on such von-Neumann machines, like artificial intelligence, search engines, neural networks, biological systems, financial analysis *etc.*, are limited by the *von Neumann bottleneck* [5]. This bottleneck results due to frequent and large amounts of data transfer between the physically separate memory units and compute cores. Moreover, frequent to-and-fro data transfers incur large energy overheads in addition to limiting the overall throughput. This has largely restricted the execution of DNNs to large-scale data-centers, due to such high power demands.

Nowadays, most real-time data is generated at the edge-devices, such as sensor nodes, drones, and IoT devices. Most of these devices are battery-operated, and thus have limited battery life. Transferring large amounts of data from the edge devices to the cloud is not only energy expensive, but sometimes undesirable due to security reasons, such as in defense or automotive applications. Thus, there is a need for processing data at the edge, to enable energy-efficient DNN inference. There have been innovations in both algorithmic as well as hardware fronts, to mitigate the energy problem of exploding DNNs. Recently, there have been emergence of *memory-friendly* quantized networks, such as binary networks [6], XNOR-Net [7], and ternary networks [8], [9]. The basic idea is to reduce the bit-precision of

the network parameters (weights, or activations, or both), from full-precision (32-bit or 64-bit floating point) to low-precision fixed-point notation (1-bit, 2-bit, etc.). This drastically reduces the computational complexity of the network, while also reducing the amount of data movement, without significant loss in state-of-the-art accuracies owing to the error-resiliency of neural networks, and their ability to re-train.

On the hardware side of things, there have been significant interest in beyond von-Neumann computing, especially the paradigm of in-memory computing [10], which aims to embed logic within the memory array in order to reduce memory-processor data transfers. In-memory techniques tend to bypass the von-Neumann bottleneck by accomplishing computations right inside the memory array. In other words, in-memory-compute blocks store data exactly like a standard memory, however, they enable additional operations without expensive area or energy overheads. By enabling logic computations in-memory, significant improvements, both in energy efficiency and throughput are expected [11]–[14]. This approach embeds some basic computations within the memory arrays, where the data is stored. By using such enhanced memory structures, frequent and unnecessary data-transfers between the memory and the compute units can be eliminated, without significantly changing the memory hierarchy and the conventional read/write functionality of memory arrays. Moreover, this opens up the internal bandwidth of memory arrays, which is much larger than the external input/output bandwidth, and can be exploited to enable parallelism.

Due to the potential impact of in-memory computing on future computing platforms, various proposals spanning right from conventional complementary metal-oxide semiconductor (CMOS) to beyond-CMOS technologies can be found in the literature. For example, Ref. [15] proposed integrating an ALU (arithmetic-logic-unit) close to the memory unit to exploit the wide memory bandwidth, while Ref. [11] reconfigures a standard 6 transistor (6T) static random-access memory (SRAM) cells as content addressable memories (CAMs) and enable bit-wise logical operations. 6T-SRAM cells have also been used to implement machine learning classifiers [16], and dot-products in analog domain for pattern recognition [13]. The underlying idea is to enable multiple rows of memory bit-cells and directly read out a voltage at the pre-charged bit-lines corresponding to the desired operation. However, the 6T-SRAM bit-cells have a coupled read-write path that imposes conflicting constraints

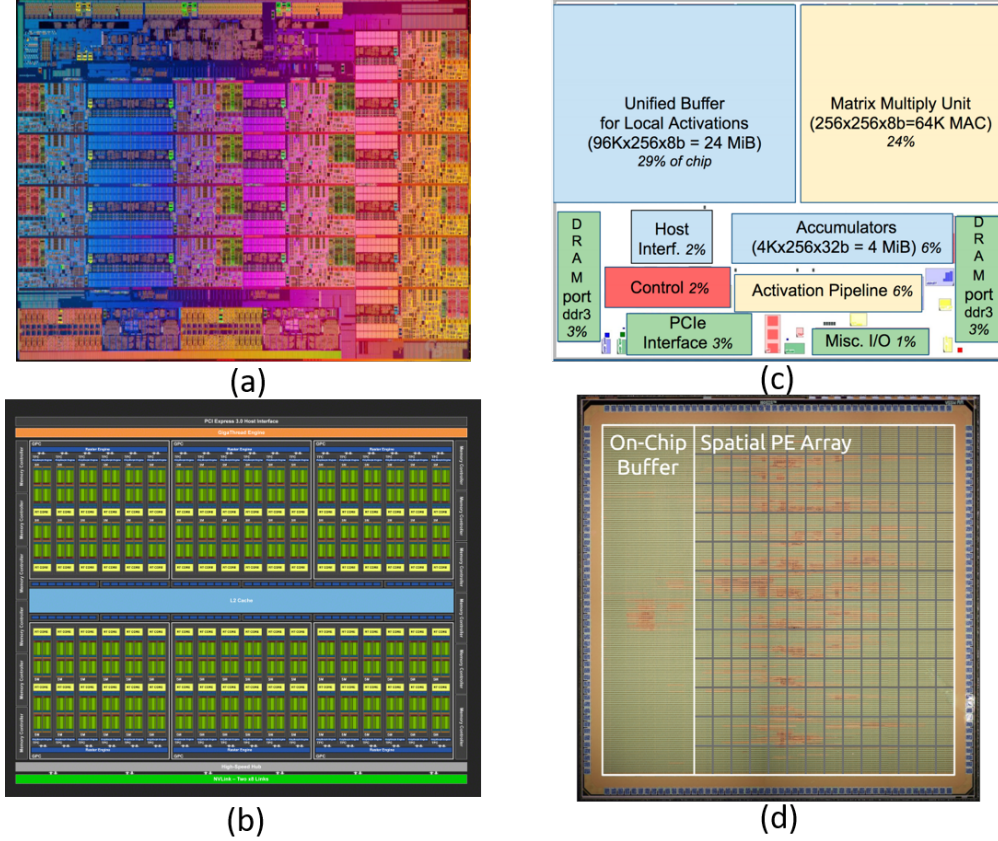


Figure 1.1. (a) Intel Xeon CPU [18]. (b) NVIDIA Turing GPU [19]. (c) Google TPU [20]. (d) Eyeriss chip [21].

on the design of the 6T cell, thereby raising issues of read-disturb failures. Moreover, activating multiple word-lines may cause short-circuit paths, thereby flipping the cell states non-deterministically. The read-disturb failure is further accentuated by the fact that once the BL has discharged, activating subsequent word-lines perform a *pseudo-write* operation on the 6T cell, given the shared read-write path. A 6T-SRAM based on the deeply depleted channel (DDC) technology [17] was recently proposed for searching and in-memory computing applications, which had decoupled read-write paths. However, all of these proposals perform the computation in the peripheral circuits and read out the data. A subsequent memory-write operation is required to store the data back in the memory array.

Looking at the die area of both standard systems (for example, CPU and GPU), as well as the domain-specific accelerators built for ML workloads (for example TPU[22] and

Eyeriss[23]), as shown in Fig. 1.1, we can observe that a significant chip area is consumed by on-chip caches, giving us significant opportunity for enhancing these memories with compute capabilities, at various levels of hierarchies. Standard on-chip caches use CMOS 6-Transistor Static Random Access Memories (6T-SRAMs) which are optimized for fast read and write operations. However, 6T-SRAMs have a shared read/write port, and are not suited for in-memory computing applications due to read-stability concerns. In 8T-SRAMs, there is an additional read port, thereby decoupling the read and write operations at the cost of two extra transistors. This provides additional SRAM stability, thereby enabling the possibility of in-memory computations [24]. Going to 9T- and 10T-SRAM cells improves the SRAM stability further, but at the cost of reduced storage density. Thus, there is a tradeoff between the in-memory computing functionality and storage density.

In addition, almost all beyond CMOS non-volatile technologies have been extensively explored for possible applications to in-memory computing [25]. These include works based on resistive RAMs [26]–[32], spin-based magnetic RAMs [33]–[39], phase change materials [40], and ferroic materials [41]. Such emerging non-volatile technologies promise denser integration, energy-efficient operations and non-volatility as compared to the CMOS based memories, and are suitable for in-memory computations [42]. However, these emerging technologies are still under extensive research and development phase and their large scale commercialization for on-chip memories is far-fetched.

2. ENABLING IN-MEMORY BOOLEAN COMPUTATIONS IN STANDARD 8T SRAM ARRAYS

2.1 Introduction

Since the invention of transistor switches [43], there has been an ever-increasing demand for speed and energy-efficiency in computing systems. Almost all the state-of-the-art computing platforms are based on the well-known *von-Neumann* architecture which is characterized by decoupled *memory storage* and *computing cores*. Running data-intensive applications on such von-Neumann machines, like artificial intelligence, search engines, neural networks, biological systems, financial analysis *etc.*, are limited by the *von Neumann bottleneck* [5]. This bottleneck results due to frequent and large amounts of data transfer between the physically separate memory units and compute cores. Moreover, frequent to-and-fro data transfers incur large energy overheads in addition to limiting the overall throughput.

In order to overcome the von-Neumann bottleneck, there have been many efforts to develop new computing paradigms. One of the most promising approach is the *in-memory computing*, which aims to embed logic within the memory array in order to reduce memory-processor data transfers. Conceptually, the in-memory compute paradigm is illustrated in Fig. 2.1. It shows two physically separated blocks – the processor and the memory unit and the associated computing bottleneck. In-memory techniques tend to bypass the von-Neumann bottleneck by accomplishing computations right inside the memory array, as shown in the figure. In other words, in-memory-compute blocks store data exactly like a standard memory, however, they enable additional operations without expensive area or energy overheads. By enabling logic computations in-memory, significant improvements, both in energy efficiency and throughput are expected [11]–[14].

Due to the potential impact of in-memory computing on future computing platforms, various proposals spanning right from conventional complementary metal-oxide semiconductor (CMOS) to beyond-CMOS technologies can be found in the literature. For example, Ref. [15] proposed integrating an ALU (arithmetic-logic-unit) close to the memory unit to exploit the wide memory bandwidth, while Ref. [11] reconfigures a standard 6 transistor (6T) static random-access memory (SRAM) cells as content addressable memories (CAMs)

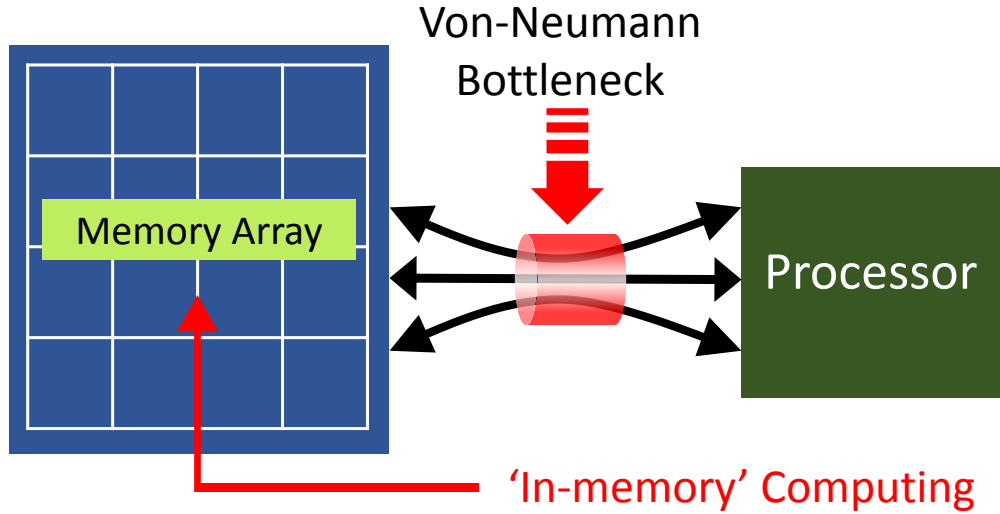


Figure 2.1. Illustration of the von-Neumann bottleneck. Frequent to-and-fro data transfers between the processor and memory units incur large energy consumption and limits the throughput. Computing within the memory array enhances the memory functionality thereby reducing the number of unnecessary transfers of data for certain class of operations like vector bit-wise Boolean logic *etc.*

and enable bit-wise logical operations. 6T-SRAM cells have also been used to implement machine learning classifiers [16], and dot-products in analog domain for pattern recognition [13]. The underlying idea is to enable multiple rows of memory bit-cells and directly read out a voltage at the pre-charged bit-lines corresponding to the desired operation. However, the 6T-SRAM bit-cells have a coupled read-write path that imposes conflicting constraints on the design of the 6T cell, thereby raising issues of read-disturb failures. Moreover, activating multiple word-lines may cause short-circuit paths, thereby flipping the cell states non-deterministically. The read-disturb failure is further accentuated by the fact that once the BL has discharged, activating subsequent word-lines perform a *pseudo-write* operation on the 6T cell, given the shared read-write path. A 6T-SRAM based on the deeply depleted channel (DDC) technology [17] was recently proposed for searching and in-memory computing applications, which had decoupled read-write paths. However, all of these proposals perform the computation in the peripheral circuits and read out the data. A subsequent memory-write operation is required to store the data back in the memory array. Thus, we

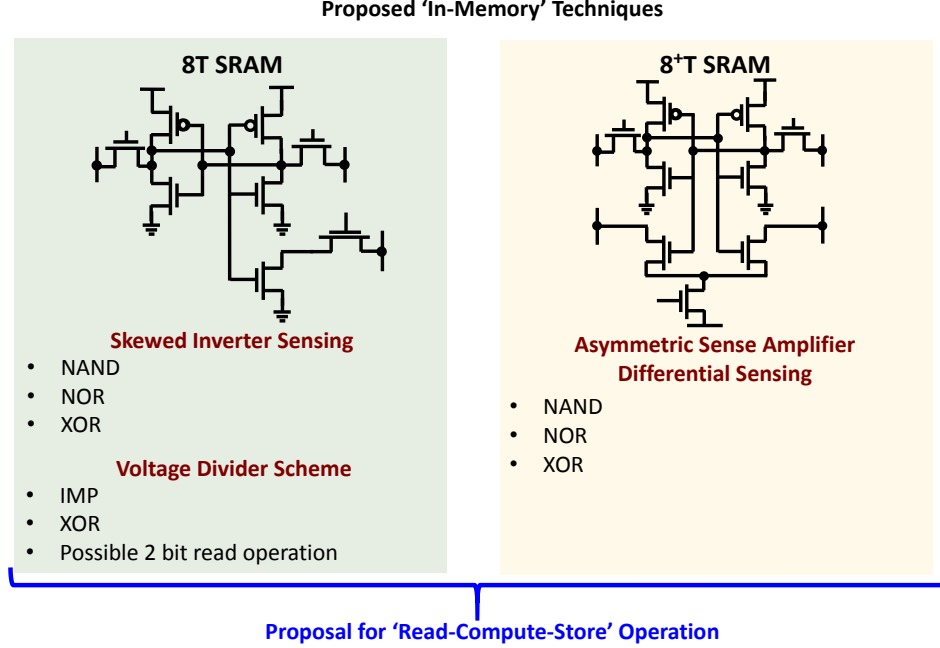


Figure 2.2. A summary of *In-Memory* computing schemes proposed. With respect to the 8T cell, we present bit-wise NAND, NOR and XOR operations using skewed inverter sensing. Further, we present the voltage-divider based operation of 8T-cells for IMP and XOR gates. With respect to the 8⁺T-cells, we present bit-wise NAND, NOR and XOR operations using asymmetric differential SAs. Moreover, a ‘*read-compute-store*’ operation has been presented for both types of bit-cells.

use standard CMOS 8T- and 8⁺T Differential SRAM cells due to their decoupled read-write mechanisms, for performing in-memory computations. Moreover, we go a step further and propose the novel ‘*read-compute-store*’ scheme, where the computed result can be stored *in-situ*, within the memory array, without the need for latching the result and performing a subsequent memory-write instruction. In addition, recently memristor like multi-bit dot product computations using 8T cells has been proposed in [44].

In addition, almost all beyond CMOS non-volatile technologies have been extensively explored for possible applications to in-memory computing [25]. These include works based on resistive RAMs [26], spin-based magnetic RAMs [33]–[35], and phase change materials [40]. Such emerging non-volatile technologies promise denser integration, energy-efficient operations and non-volatility as compared to the CMOS based memories, and are suitable for

in-memory computations [42]. However, these emerging technologies are still under extensive research and development phase and their large scale commercialization for on-chip memories is far-fetched.

We explore in-memory vector operations in *standard* CMOS 8T- and 8⁺T Differential SRAM cells with minimal modifications in the peripheral circuitry. We call the augmented version of the SRAM bit-cells with extra in-memory compute features as *the X-SRAM* [24]. We propose *at least six different techniques* to enable Boolean computations. The 8T and 8⁺T cells lend themselves easily for enabling in-memory computations because of the following three factors. 1) The read ports of the 8T and 8⁺T cells are isolated and can be easily configured to enable in-memory operations. 2) Also, in sharp contrast to the 6T cells, 8T and 8⁺T cells do not suffer from read disturb and hence multiple read word-lines within the memory array can be simultaneously activated. 3) In addition, in this manuscript, we exploit the two port structure of the 8T and 8⁺T cells to propose a novel *read-compute-store* operation, wherein, the computed Boolean data can be stored into the memory array without actually latching the data followed by a subsequent memory write-operation. Later in Appendix, we describe the in-memory computations in standard 6T-SRAMs using the staggered activation of word-lines, as was presented for analog computing in Ref. [13].

Some of the key highlights in comparison to previous works are enumerated below.

1. We firstly leverage the fact that two simultaneously activated read-word-lines for the standard 8T cells are inherently ‘wire NORed’ through the read bit-line. By using a skewed inverter at the sensing output, we demonstrate that NOR operation can be easily achieved. Further, we also show that NAND logic can similarly be accomplished using another skewed inverter. Note, unlike 6T cells, simultaneous activations of two read word-lines do not impose any read-disturb concerns, thereby opening up a wider design space for optimization.
2. Further, by applying appropriate voltages, we show that two activated read ports of the 8T cell can be configured as a voltage divider. Based on such *voltage divider scheme* we present in-memory vector IMP as well as XOR logic gates. The voltage

divider scheme not only allows in-memory computations, but also augments the read mechanism by allowing a possible two bit-read operation under specific conditions.

3. Subsequently, we also present in-memory NAND and NOR computations (along with XOR) in the recently proposed 8^+ T cells [45], using *asymmetric sense amplifiers* (SA). The 8^+ T cells are more robust since they allow differential read sensing as opposed to the standard 8T cells that are characterized by single ended sensing. The usual memory read/write functionality of the SRAM cell is not disturbed due to the use of asymmetric sense amplifiers. We also show that the same hardware, including the SA, can be shared for an in-memory operation and also for the normal memory read operation. Moreover, the extra hardware enhances the memory read operation, by acting as a check for read failures.
4. We propose a novel ‘*read-compute-store*’ scheme for the 8T and 8^+ T bit-cells, wherein the computed data can directly be written into the desired memory location, without having to latch the output and perform a subsequent memory write operation. This exploits the decoupled read-write paths of the 8T and 8^+ T bit-cells.
5. We perform Monte-Carlo simulations including voltage and temperature variations to verify the robustness of the proposed in-memory operations for the 8T and the 8^+ T bit-cells. Energy, delay and area numbers have been presented for each of the proposed scheme.
6. We demonstrate the effectiveness of using in-memory bitwise computations in a typical von-Neumann machine, wherein the conventional SRAM is replaced by the proposed X-SRAM for Advanced Encryption Standard (AES) algorithm. Our system level simulations indicates 75% reduction in memory accesses thereby saving energy expensive data transfers.

2.2 In-Memory Computations in 8-Transistor SRAM Bit-Cells

As discussed in the introduction, 8T cells have favorable bit-cell structure to enable in-memory computing. Specifically we would exploit the isolated read mechanism and the two

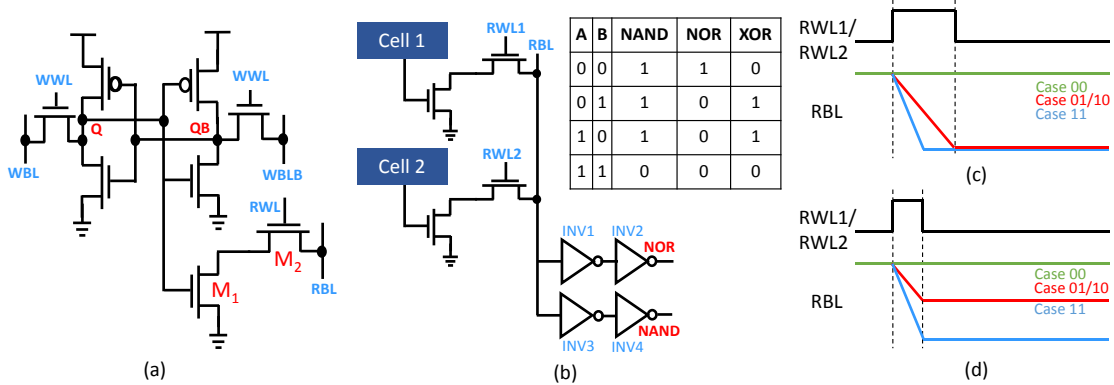


Figure 2.3. a) Schematic of a standard 8T-SRAM bit-cell. In addition to the standard 6T cell, two additional transistors form the read path using a separate read bit-line (RBL). b) Single ended sensing of NAND/NOR using gated skewed inverters. Figure also shows the truth table for NAND/NOR/XOR operations. c) Timing diagram for reading NOR output of Cell 1 and Cell 2. d) Timing diagram for reading NAND output of Cell 1 and Cell 2.

port cell topology to embed NAND, NOR, IMP and XOR logic within the memory array. Further, by leveraging the separate read and write ports of the 8T cell, we also propose a ‘*read-compute-store*’ scheme, wherein, by minimal changes in the peripheral circuits, the computed Boolean results can be stored in the desired row of the memory array in the same cycle without the need of latching the results and performing a subsequent write operation.

For each proposal, we first describe the circuit operation using representative illustrations of the transient waveforms followed by actual SPICE based transient simulations under Monte-Carlo analysis. Further, we also present a distribution graph for the key voltages that represent worst case scenarios including temperature as well as voltage variations. Note, in general global process variations can be taken care by proper calibrations, therefore, we concentrate on intra-die threshold voltage variation along with variations in temperature and supply voltage. Towards the end of the manuscript, we tabulate the pros-and-cons of the proposed techniques in a comparative manner.

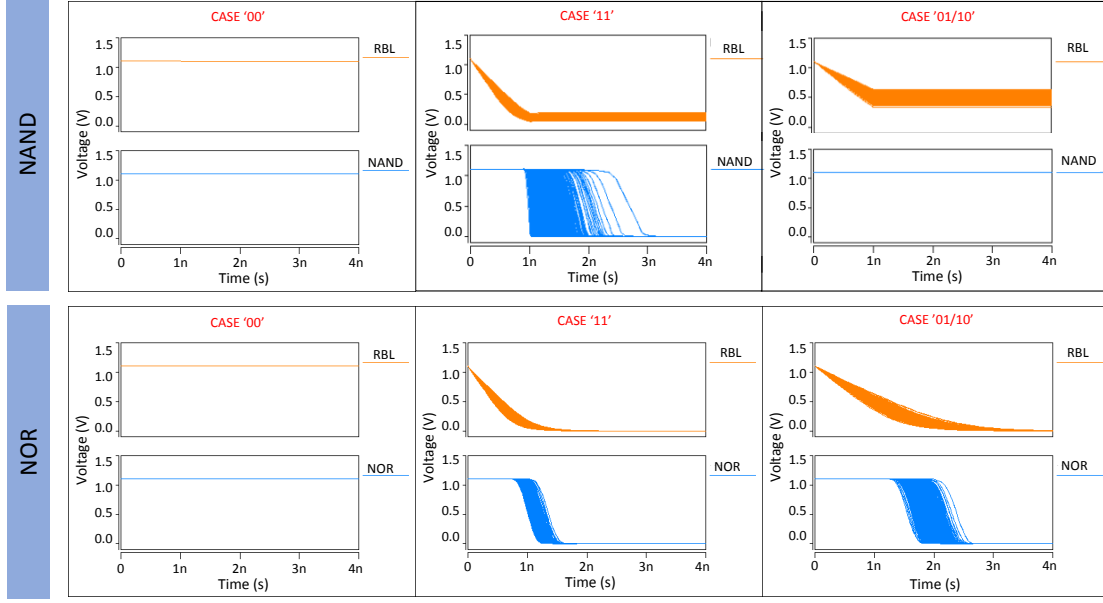


Figure 2.4. Monte-Carlo simulations in SPICE for NAND and NOR outputs for all possible input cases – ‘00,01,10,11’, in presence of 30mV sigma variations in threshold voltage.

2.2.1 8-Transistor SRAM: NOR operation

The 8T SRAM cell is shown in Fig. 2.3(a). It consists of the usual 6T cell augmented by additional read port constituted by transistors M1-M2. The write operation is similar to the 6T cell, whereas for the read operation, RWL is activated (WWL is low). The RBL is initially pre-charged and if $Q = '1'$ the RBL discharges otherwise it stays at its initial precharged condition. This decoupled read port for the 8T cell allows to have large voltage swing (almost rail-to-rail) on the RBL during the read operation without any concerns of read disturb failure.

The output of a NOR operation is ‘1’ only if both the inputs are ‘0’. Consider we activate two RWLs corresponding to the rows storing vector operand ‘A’ and vector operand ‘B’, respectively, as shown in Fig. 2.3(b). Due to the decoupled read ports, both the RWLs can be activated simultaneously without any read disturb concerns as opposed to the 6T cell. The precharged RBL line retains its precharged state if and only if both the bits Q corresponding to operands ‘A’ and ‘B’ are ‘0’. In other words, as shown in Fig. 2.3(c) RBL remains high only if $Q = '0'$ for both ‘A’ and ‘B’. Thus, merely by activating the two RWLs,

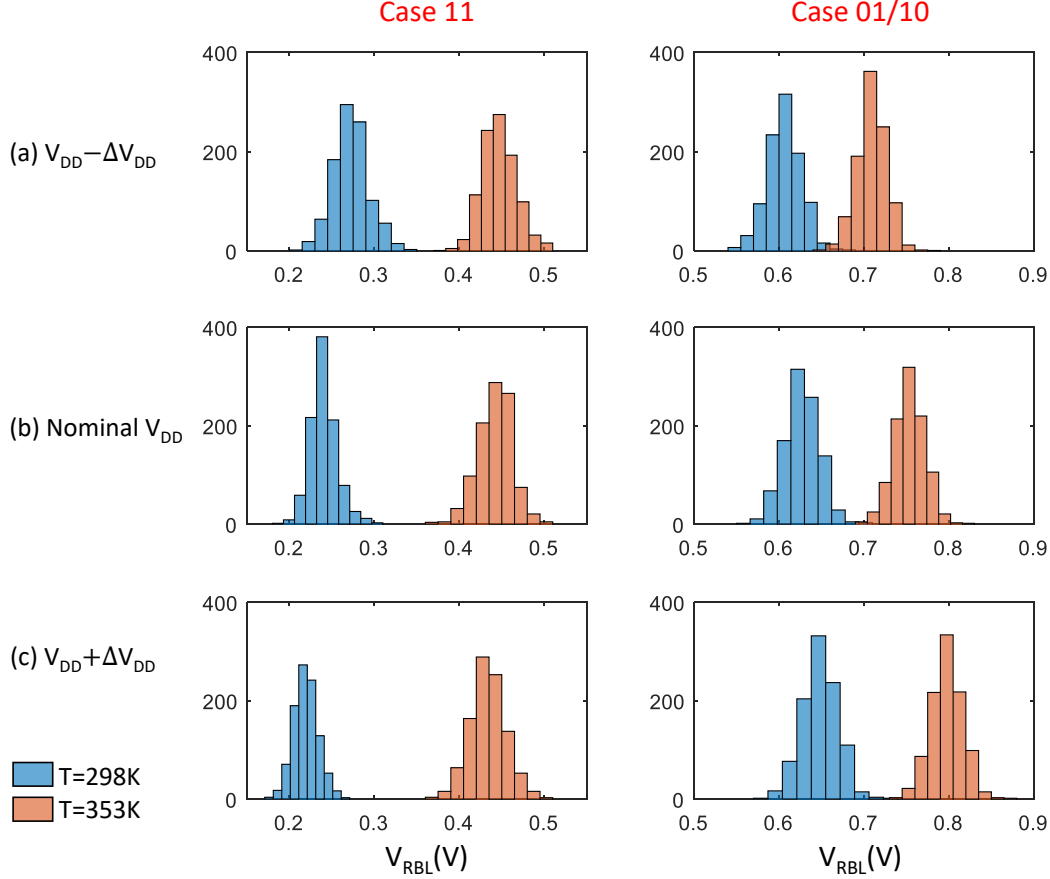


Figure 2.5. Monte-Carlo simulations across process corners (TT corner and SS corner shown) under voltage and temperature variations for NAND outputs for the borderline cases – ‘01/10’ and ‘11’. The distribution of RBL voltage is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD} .

data stored in the two bit-cells are ‘wire NORed’. A gated inverter (INV1) is connected to the RBL such that the inverter output goes low if the RBL remains high. Thereby, the output of the cascaded inverter (INV2) mimics the NOR operation. Note, the NOR operation is same as the usual read operation except that we have turned ON two RWLs instead of one. Thus, NOR can be easily achieved in the 8T bit-cell without any significant overhead. The timing diagram for the NOR operation is shown in Fig. 2.3(c). It is also interesting to observe that although we have discussed the NOR operation for two inputs, the proposed scheme can in fact be extended to n -input NOR operations. For an n -input NOR operation

n -read word-lines can be simultaneously activated and RBL would remain high only if all the corresponding operands are ‘0’ which would represent the n -input NOR truth table.

2.2.2 8-Transistor SRAM: NAND operation

Let us consider that we activate two RWLs corresponding to vector operands ‘A’ and ‘B’, respectively. The precharged RBL will eventually go to 0V if Q for any one of the input operand is ‘1’. However, the fall time of the signal at RBL from the precharged value to 0V would depend strongly on the fact, whether any one Q is high or if both the Q bits are high simultaneously. In other words, only if both the Qs are ‘1’, the discharge of the precharged RBL line would be fast enough. In Fig. 2.3(d), we have shown schematically the state of the RBL for all input cases. In order to exploit the different discharge rates of the RBL, the RWL signal had to be timed such that the RBL does not discharge completely in cases ‘01/10’. This allows a difference in voltage levels on RBL in the two cases (‘01/10’ and ‘11’). The trip point of the inverter INV3 is chosen such that it goes high only for the case ‘11’, thus output of inverter INV4 mimics the NAND operation.

Fig. 2.4 shows the SPICE transient simulation for the NAND and NOR proposals, under 30mV sigma threshold voltage variation in transistors. We used 45-nm Predictive Technology Models (PTM) [46] for simulating the circuits. A BL and BLB capacitance of 10fF was assumed for all the simulations. As discussed earlier, the NAND computation has a narrower design margin due to its timing critical operation as opposed to the NOR logic. Specifically, for the NAND operation a discharge path with two parallel transistors needs to be distinguished from the discharge path with one transistor. To analyze the robustness and the design margin, we performed a rigorous variation analysis across process corners including voltage and temperature variations for the NAND operation, as shown in Fig. 2.5. Monte-Carlo simulations with 30mV sigma threshold voltage variation were performed along with a $\pm 10\%$ variation in nominal V_{DD} (ΔV_{DD}). The simulations were repeated for two different temperatures. The figure shows the resultant distribution of voltage on RBL for the borderline cases ‘11’ and ‘01/10’, at the instant when RWL is pulled LOW.

In order to study the effect of variations due to different process corners, we also performed simulations assuming global variations in the threshold voltage, the simulations were performed for all possible corners including SS (slow NMOS, Slow PMOS), SF (Slow NMOS, Fast PMOS), FS (Fast NMOS, Slow PMOS) and FF (Fast NMOS, Fast PMOS). The threshold voltages for respective corners were globally shifted in appropriate directions for each of the process corners for both the PMOS and the NMOS transistors. For example, for the SS and FF corners, the threshold voltages were increased or decreased by $\sim 90\text{mV}$ to imitate the affect of process corners. These global shifts in threshold voltages were then super-imposed by random VT variations to evaluate the cumulative effect. In Fig. 5, we have shown the Monte-Carlo results for two different process corners – the nominal case (TT) and for the SS corner, for two different temperatures including $\pm 10\%$ variation in supply voltage. Note, similar results were obtained for other process corners as well, however to avoid clutter, we have shown two representative results for the process corners. It can be observed, we obtain a 50mV worst cases sense margin in Fig. 2.5(a) for a -10% nominal V_{DD} . Also, the timing for the NAND operation can be controlled by a digitally programmable delay based control signal for tuning the pulse activation of the RWL [47]. Such a programmable delay path would require a one-time calibration depending on the process corner, for proper functionality.

In addition to the NAND and NOR operations, by NORing the outputs of the AND (INV3) and the NOR (INV2) gates together, XOR operation can be easily achieved. In summary, we have shown that the very bit-cell topology of the 8T cell can be exploited to accomplish in-memory NOR, NAND and XOR computations. In the next sub-section, we would discuss another proposal for embedding IMP as well as XOR gate within the 8T SRAM array by utilizing the proposed voltage divider scheme.

2.2.3 8 Transistor SRAM: Voltage Divider Scheme for IMP and XOR gates

In this sub-section, we present a method of implementing IMP and XOR operation using 8T cell by exploiting the voltage divider principle. Let us consider, the circuit shown in Fig. 2.6(a). Let us assume the first operand is stored in the upper bit-cell corresponding to the

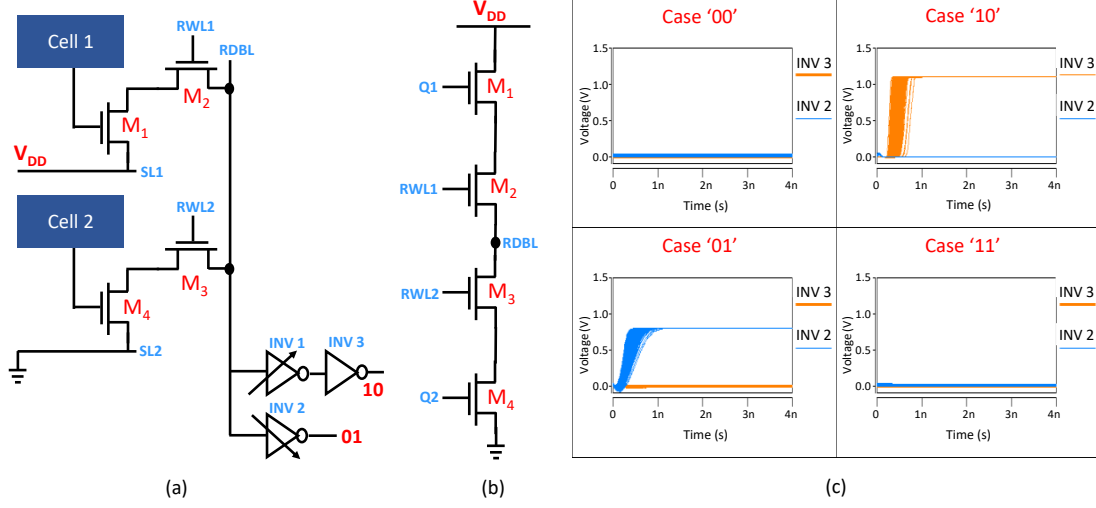


Figure 2.6. a) Circuit schematic of the 8T-SRAM for implementing the voltage-divider scheme. b) Equivalent circuit traced by transistors $M_1 - M_4$ while data is read from Cell 1 and Cell 2. c) Monte-Carlo simulations in SPICE for all possible input cases, showing the output of the two asymmetric inverters.

line $RWL1$, while the second operand is stored in the lower bit-cell corresponding to $RWL2$. In the conventional 8T cell, the source of transistors M_1 and M_4 are connected to ground. In the presented circuit, the source of the transistors M_1 and M_4 are connected to respective source lines ($SL1$ and $SL2$ shared along respective rows). During the normal operations, the SLs can be grounded, thereby accomplishing usual 8T SRAM read and write operations.

During the in-memory computation mode, the $SL1$ is pulled to V_{DD} , while the $SL2$ is grounded. $RWL1$ and $RWL2$ are initially grounded and $RDBL$ is pre-charged to a voltage V_{pre} (chosen to be 400mV). After the pre-charge phase, transistors M_2 and M_4 are switched ON, thereby $M_1 - M_2 - M_3 - M_4$ form a voltage divider and $RDBL$ forms the middle node of the voltage divider structure (see Fig. 2.6(b)). Note, in the voltage divider configuration, M_1 and M_2 are strongly source degenerated. In order to make sure M_1 and M_2 are sufficiently ON, we boosted the V_{DD} of ‘Cell 1’ and $RWL1$ such that the gate of M_1 and M_2 have enough overdrive when the ‘Cell 1’ is storing a digital ‘1’ ($Q = ‘1’$ and $QB = ‘0’$).

In the voltage divider configuration $M_1 - M_2 - M_3 - M_4$, $RDBL$ retains its precharged voltage V_{pre} if both the bit-cells are storing digital ‘0’ (*i.e.* M_1 and M_4 are OFF). Similarly,

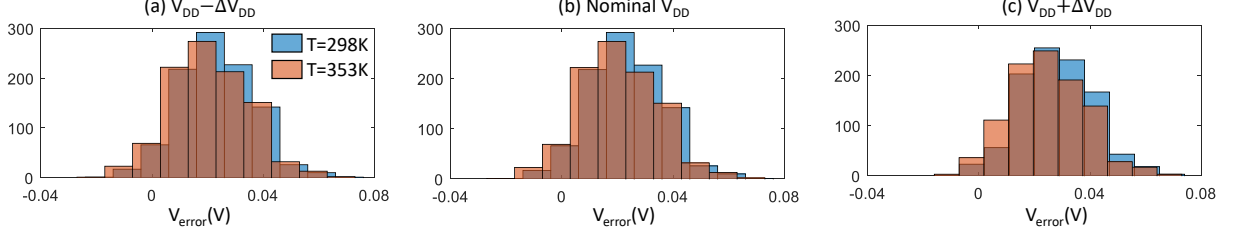


Figure 2.7. Monte-Carlo simulations with variations in supply voltage and temperature across process corners for the voltage-divider scheme for the case (1,1). V_{error} is defined as the difference between the RBL voltage (when both the operands are ‘1’) and the initial pre-charge voltage V_{pre} . The distribution of V_{error} is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD} . The variations in V_{pre} are also accounted for.

if both the cells are storing a digital ‘1’ (*i.e.* M_1 and M_4 are ON), the voltage at RDBL stays close to its precharged value (400mV) due to the voltage divider effect. Thus, when the cells store (0,0) or (1,1) (where the first (second) number in the bracket indicates the data stored in Cell 1 (2)), the voltage at RDBL stays close to the precharged voltage. On the other hand, if the data stored is (1,0), then M_1 is ON while M_4 is OFF. As such, RDBL will charge to V_{DD} through transistors M_1 and M_2 . In contrast, if the data stored is (0,1), M_4 is ON while M_1 is OFF. Therefore, RDBL will discharge to 0V through transistors M_3 and M_4 . In summary, the voltage on RDBL stays close to V_{pre} when both the cells store same data. RDBL charges to V_{DD} for data (1,0) and discharges to 0V for data (0,1).

The state of the data stored in the two cells can be sensed through two skewed inverters. INV2 is skewed such that it goes high only when RDBL is much lower than V_{pre} and is close to 0V, while INV1 is skewed so that it goes low only when RDBL is higher than V_{pre} and is close to V_{DD} . In other words, high output at INV2 indicates data (0,1) while high output at INV3 indicates data (1,0). Interestingly, INV1 implements ‘A IMP B’. By ORing the output of INV2 and INV3 we can obtain the XOR of inputs A and B.

Fig. 2.7 shows the distribution of V_{error} (defined below) under V_T variations in addition to variations in temperature and supply-voltage. Monte-Carlo simulations across process corners, similar to the ones performed for NAND in the previous sub-section were performed in this case. Note, when either of the two operands Q1 or Q1 is low, the circuit in Fig.

2.6(b) reduces to an RC charging or discharging circuit, respectively. As such, if any of Q1 or Q2 is low, the RDBL would either charge up to V_{DD} or discharge to ground even under variations. The critical case arises when both Q1 and Q2 are low or high, simultaneously. For robust operation, ideally we want the RDBL voltage to stay at V_{pre} for both the cases (Q1 = Q2 = low or Q1 = Q2 = high). Therefore, we analyze the difference in voltages on the RDBL in the two cases ‘Q1 = Q2 = low’ versus ‘Q1 = Q2 = high’. We define V_{error} as the difference between the RDBL voltage for case (1,1) and case (0,0). In other words, V_{error} denotes the variation of RDBL voltage when the voltage divider is active with respect to V_{pre} . The variations in V_{pre} are also considered in the Monte-Carlo simulations. We observe that V_{error} is close to zero, making this configuration robust to variations, as shown in Fig. 7. Intuitively, the robustness of the proposed scheme stems from the fact that changes in voltage and temperature affects all the four transistors of Fig. 2.6(b) in similar manner thereby reducing any variations in the voltage at node RDBL. Moreover, since we use the static voltage developed at RDBL, unlike the time-sensitive discharge in the earlier scheme, the voltage-divider scheme is robust to process corners as well. The voltage at RDBL depends on the relative strengths of the four transistors. Since process corners induce global V_T shifts, all NMOS transistors are equally affected, making the voltage-divider ratio largely unaffected. This is evident from the two representative process corner simulations shown in Fig. 7.

Some key features of the voltage divider logic scheme are, 1) IMP is a universal gate and hence any arbitrary Boolean function can be implemented using the proposed scheme 2) if any one of the inverter outputs (INV2 or INV3) are high, it indicates the data stored is (0,1) or (1,0), thereby allowing a two bit-read operation in addition to the desired in-memory computation. However, if none of the inverters are high then a subsequent read operation would be required to ascertain if the stored data is (0,0) or (1,1). As such, in 50% cases when the data stored is (0,1) or (1,0), we can accomplish a two bit read operation, along with the in-memory compute operation.

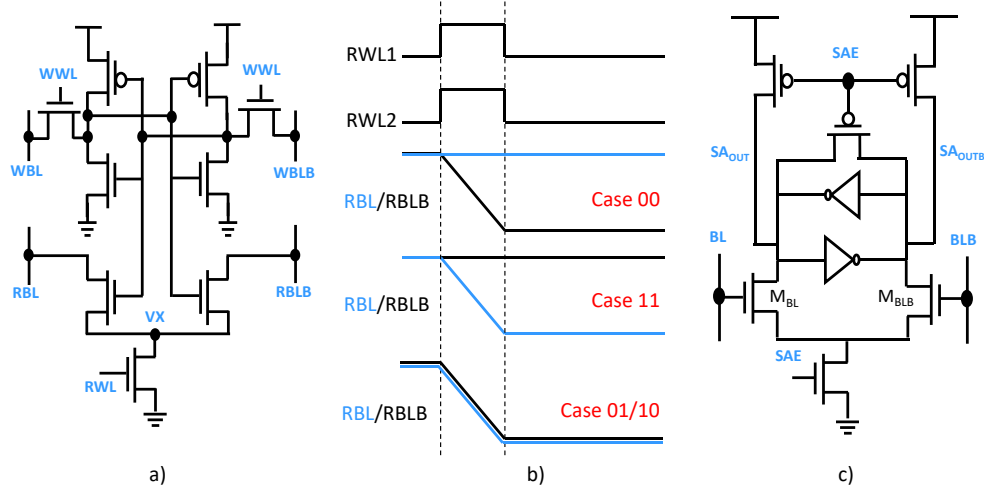


Figure 2.9. a) Circuit schematic of an 8⁺T Differential SRAM bit-cell [45]. b) Timing diagram used for in-memory computations on the 8⁺T Differential SRAM. c) Circuit schematic of the proposed asymmetric differential sense amplifier.

in Fig. 2.8(a), two read lines RWL1 and RWL2 would be activated, the compute block, which basically is the abstracted view of the skewed inverters of Fig. 2.3(b), would perform the logic computation. Now, since the read and write port for 8T cell are decoupled we can simultaneously activate a third WL, in this case the write word-line (WWL3). The computed output can be selected through a multiplexer and fed to the write drivers for directly storing the Boolean result in the bit-cells corresponding to WWL3. Thus, the fact that 8T cells have decoupled read-write ports can be leveraged to accomplish the proposed ‘read-compute-store’ scheme. Fig. 2.8(b) shows schematically the array level block diagram where the three word-lines RWL1, RWL2 and WWL3 are activated simultaneously. In Fig. 2.8(c) we show the Monte-Carlo results for storing the computed NAND output into Cell3. Note that a ‘copy’ operation can also be performed using the RCS scheme, by activating the RWL of the source row and WWL of the destination row. In this case, the input to the RCS block will simply be the SA output, which corresponds to the data stored in the bit-cells of the source row.

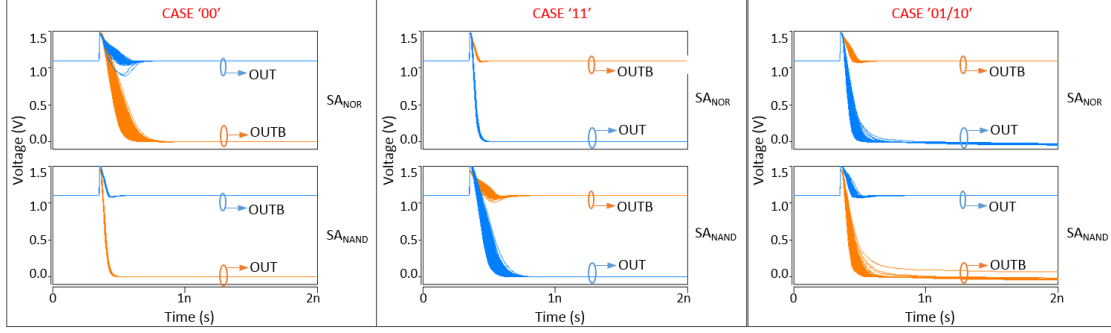


Figure 2.10. Monte-Carlo simulations in SPICE for SA outputs for all possible input cases – ‘00,01,10,11’, in presence of 30mV sigma variations in threshold voltage.

2.3 8^+ Transistor Differential Read SRAM

Recently, an 8^+ T Differential SRAM design was proposed in [45] to overcome the single ended sensing of the conventional 8T-SRAM cell. 8^+ T Differential SRAM has decoupled read-write paths with an added advantage of a differential read mechanism through the read bit-lines RBL/RBLB (see Fig. 2.9(a)), as opposed to the single-ended read mechanism of 8T-SRAM. The ninth transistor, whose gate is connected to RWL in Fig. 2.9(a) is shared by all the bit cells in the same row. The differential read operation is very similar to the read operation of a standard 6T-SRAM. The usual memory read operation is performed by pre-charging the bit-lines (RBL and RBLB) to V_{DD} , and subsequently enabling the word-line corresponding to the row to be read out. Depending on whether the bit-cell stores ‘1’ or ‘0’, RBL or RBLB discharges. The difference in voltages on RBL and RBLB is sensed using a differential sense amplifier.

Let us consider words ‘A’ and ‘B’ stored in two rows of the memory array. Note that we can simultaneously enable the two corresponding RWLs without worrying about read-disturbs, since the bit-cell has decoupled read-write paths. The RBL/RBLB are pre-charged to V_{DD} . For the case ‘AB’=‘00’ (‘11’), RBL (RBLB) discharges to 0V, but RBLB (RBL) remains in the precharged state. However, for cases ‘10’ and ‘01’, both RBL and RBLB discharge simultaneously. The four cases are summarized in Fig. 2.9(b).

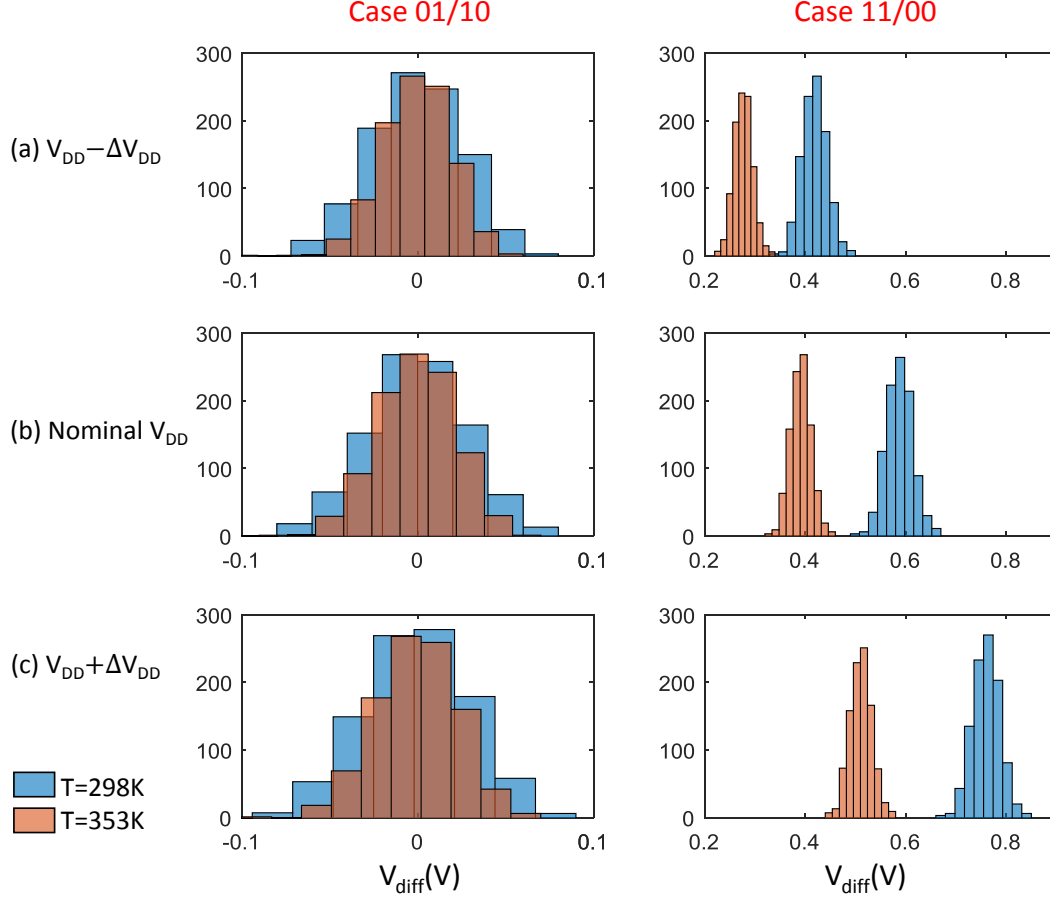


Figure 2.11. Monte-Carlo simulations across process corners under V_T and temperature and supply-voltage variations for the 8^+T SRAM configuration for the cases ‘01/10’ and ‘11/00’. V_{diff} is defined as the absolute difference between the RBL and RBLB voltages at the instant when the sense amplifier is enabled. The distribution of V_{diff} is plotted under 30mV sigma threshold voltage variations for two different temperatures and $\pm 10\%$ variation in nominal V_{DD} .

Now, in order to sense bit-wise NAND and NOR operation of ‘A’ and ‘B’, we propose an asymmetric SA (see Fig. 2.9(c)), by skewing one of the transistors. Skewing the transistors can be done in multiple ways, for example, transistor sizing, threshold voltage, body bias *etc.* In Fig. 2.9(c), if the transistor M_{BL} is deliberately sized bigger compared to M_{BLB} , its current carrying capability increases. For cases ‘01’ and ‘10’, both RBL and RBLB discharge simultaneously. However, since the current carrying capability of M_{BL} is more than M_{BLB} , SA_{out} node discharges faster, and the cross-coupled inverter pair of the SA stabilizes with

Table 2.1. Summary of proposals described in the manuscript. The table shows average energy consumption per-bit and latency for the in-memory operations on various bit-cells. Pros and cons of each proposal are also listed.

Bit-Cell	Operations	Latency (ns)	Avg. Energy/Bit (fJ)	Pros	Cons
8T-SRAM	<ul style="list-style-type: none"> NAND NOR XOR RCS 	3	17.25	<ul style="list-style-type: none"> NOR operation is very robust and can be seamlessly extended to more than two operands. Uses simple skewed inverter based sensing. 	<ul style="list-style-type: none"> Requires timing control for NAND operation. Low sense margin for NAND operation.
8T-SRAM (Voltage Divider)	<ul style="list-style-type: none"> IMP XOR RCS 	1	11.22	<ul style="list-style-type: none"> Better robustness towards global variations including voltage and temperature since global variation affects both branches of the voltage divider in similar fashion. Static design since the critical functionality is based on a stable voltage dictated by the voltage-divider effect. Possible 2 bit read operation. 	<ul style="list-style-type: none"> Requires voltage boosting for proper functionality. V_{pre} for logic functionality is different from V_{DD}.
8 ⁺ T-SRAM (Differential Cell)	<ul style="list-style-type: none"> NAND NOR XOR RCS 	1	29.67	<ul style="list-style-type: none"> Differential operation and hence improved robustness with respect to global variations including temperature and voltage. The two sense amplifiers can also be used as a sanity check for read operation. 	<ul style="list-style-type: none"> Requires two skewed sense amplifiers.

$SA_{out}=‘0’$. For the case ‘11’, RBL starts to discharge, while RBLB is at V_{DD} . The SA amplifies the voltage difference between RBL and RBLB, resulting in $SA_{out}=‘1’$. Whereas for the case ‘00’, RBLB starts to discharge, while RBL is at V_{DD} , giving $SA_{out}=‘0’$.

Thus it can be observed that SA_{out} generates an AND gate (thus, SA_{outb} outputs NAND gate). Similarly, by sizing the M_{BLB} bigger than M_{BL} , OR/NOR gates can be obtained at the SA outputs. Finally, two SAs in parallel (one with M_{BL} up-sized, SA_{NAND} , and one with M_{BLB} up-sized, SA_{NOR}) enable bit-wise AND/NAND and OR/NOR logic gates. Moreover, an XOR gate can be obtained by combining the AND/NAND and OR/NOR outputs using an additional NOR gate. Thus, in a single memory read cycle, we obtain a class of Boolean bitwise operations, read directly from the asymmetrically sized SA outputs. SPICE transient simulations with 30mV sigma variations in the threshold voltage for all input data cases are summarized in Fig. 2.10. Monte-carlo analysis across process corners with V_T variations and variations in supply voltage and temperature for the 8⁺T Differential SRAM are shown in Fig. 2.11. V_{diff} is defined as the absolute difference between the RBL and RBLB voltages at

the instant when the sense amplifier is enabled. For the case ‘01/10’, V_{diff} should be close to 0V to allow the asymmetry in the SA to determine the output. Whereas, for the case ‘11/00’, V_{diff} should be large enough, so that the output is driven by the differential voltage difference between RBL and RBLB, and not due to the asymmetry in SA. The difference in RBL and RBLB voltages for various cases is shown in Fig. 2.11. Due to global V_T variations across process corners, the discharge on both RBL/RBLB is affected in the same manner. Moreover, since we use a differential voltage sense-amplifier, these global variations cancel, thereby making this scheme robust to process corners. Note, even in worst case the voltage difference between the ‘01/10’ and ‘11/00’ case is sufficient for proper differential SA operation. Interestingly, this difference also increases with increase in V_{DD} , hence voltage boosting can be easily employed to increase the design margin.

It is worthwhile to note that the two SAs can be used for regular memory read operations as well. The two cases of a typical memory read operation are similar to the cases ‘11’ and ‘00’ in Fig. 2.9(b). Both SAs will generate the same output corresponding to the bit stored in the cell. Moreover, the output of the XOR gate inherently acts as an in-memory check for possible read failures. The RCS scheme described in Section 2.2 can also be applied to 8⁺T Differential SRAMs due to decoupled read-write paths. Along with the two RWLs from where the input operands are read, a WWL can also be enabled which would eventually store the Boolean output within the memory array in the same cycle.

Using 8⁺T cells is advantageous over the conventional 8T cells for in-memory bit-wise logic operations because of better robustness due to the differential read operation, in contrast to the single ended read in 8T-SRAM cells.

2.4 Discussions

In sections II and III, we have seen various ways of implementing basic Boolean operations using the 8T and the 8⁺T bit-cells. Table 2.1 presents the average energy per-bit and latency for each of the proposed in-memory compute techniques. The 8T cell allows separate read write ports, thereby alleviating any possible read-disturb failure concerns. In addition, it

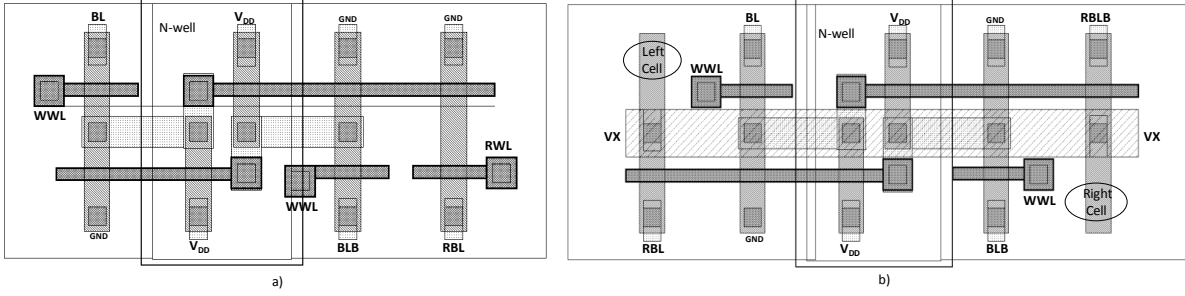


Figure 2.12. a) Thin cell layout for the standard 8T-SRAM bit-cell shown in Fig. 2.3(a). b) Thin cell layout for the 8⁺T Differential SRAM bit-cell [45] illustrated in Fig. 2.9(a). Left- and right-most diffusion tracks are shared with adjacent bit-cells. The ninth transistor in Fig. 2.9(a) is common for the row and is connected at the periphery to the node ‘VX’.

also supports the proposed RCS scheme. However, 8T cell suffers from robustness concerns due to its single ended sensing.

Using the 8⁺T cell, on the other hand, allows differential sensing like the conventional 6T cell, while also allowing separate read and write ports. It thus combines the benefits of both the standard 6T and the 8T cells. The thin cell layout of standard 8T bit-cells and the 8⁺T bit-cells are shown in Fig. 2.12. Standard 8T cell requires five diffusion tracks, while the 8⁺T cell requires six. However, left- and right-most diffusion tracks are shared with adjacent bit-cells, thereby achieving similar area per-bit as compared to the standard 8T cell [45].

Note, since the differential read scheme for the 8⁺T cell is functionally similar to the conventional 6T cell, NOR and NAND gates (along with the XOR gate) can also be implemented in the 6T based memory array. However, due to the shared read-write paths of the 6T cell, the word-lines cannot be simultaneously activated and require a sequential activation. In addition, 6T cells are read disturb prone and hence would exhibit much lesser robustness than the proposed 8T and 8⁺T cells. Nevertheless, in the Appendix we have included a description of how the 6T cells can be used to accomplish NOR, NAND and XOR operations. We also show that an in-memory ‘copy’ operation can also be easily achieved in the 6T cell due to its shared read/write paths.

Finally, it is worth noting that although we have proposed multiple in-memory techniques in this manuscript, the choice of the bit-cell and the associated Boolean function would heav-

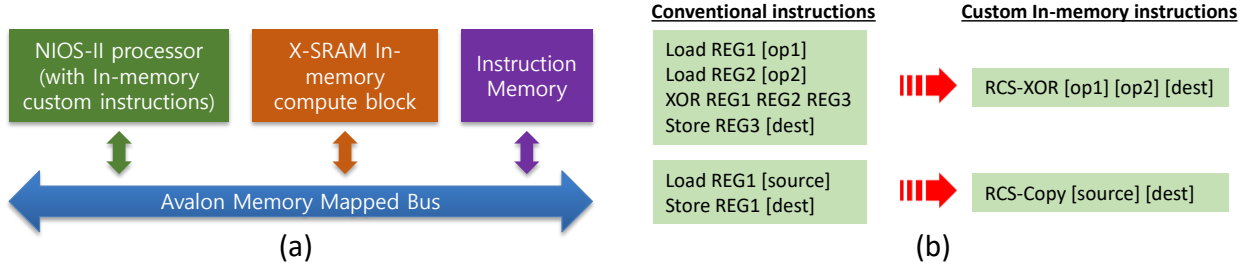


Figure 2.13. (a) System-level implementation of a typical von-Neumann architecture with X-SRAM as the memory block. The processor, data-memory and the instruction-memory blocks are connected via a shared system bus. (b) Illustration of custom in-memory instructions added to the instruction set of the Nios-II processor. Substituting in-memory instructions reduces unnecessary read-writes into the memory.

ily depend on the target application. Thus, Table 2.1 summarizes the pros and cons of each proposal. The aim of the present manuscript is to demonstrate various possible techniques that can be utilized in conventional CMOS based memories for accomplishing in-memory Boolean computations. Since the present proposal augments the functionality of the memory arrays without changing the basic circuitry, it has wide applications in diverse computing systems, few of them are – 1. A standard von-Neumann general-purpose processor with SRAM replaced by X-SRAM. 2. A modified GPU, wherein the SRAM based register files are replaced by X-SRAM arrays. 3. A machine learning or artificial intelligence processor, for example, a binary neural network accelerator. As an example, in the next section we would present an encryption accelerator using the proposed X-SRAM.

2.5 X-SRAM based non-standard von-Neumann Computing for AES Encryption

In this section, we evaluate the system-level implications of using X-SRAMs instead of conventional SRAMs as the memory blocks in a typical von-Neumann based architecture taking advanced encryption algorithm (AES) as a case study. X-SRAMs enable extra functionalities within the memory block, as described in previous sections, through massively parallel vector Boolean operations. By utilizing such *in-memory computations*, we expect

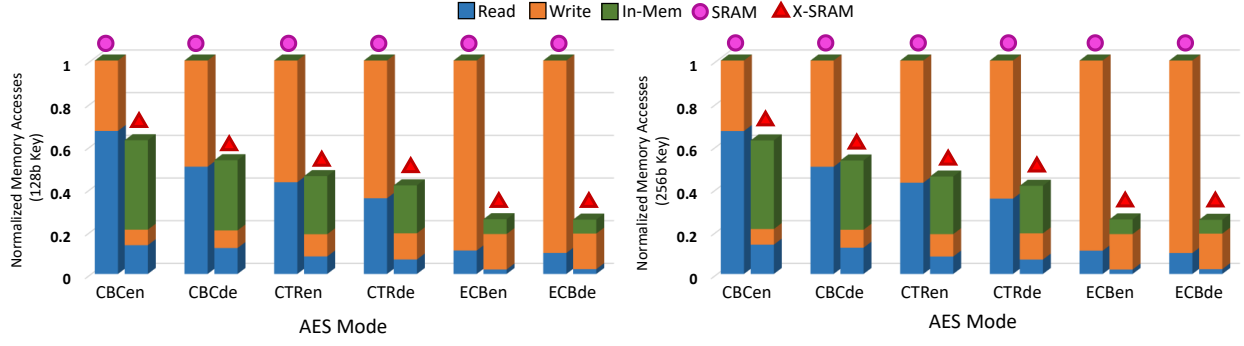


Figure 2.14. Normalized number of memory accesses for various AES encryption and decryption modes and two different key-sizes, with and without using X-SRAM custom in-memory instructions. The total memory transactions are split into memory read instructions, memory write instructions and custom in-memory instructions.

reduction in energy expensive data movements over the bus between the processor and the memory blocks.

2.5.1 Simulation Methodology

A typical von-Neumann system implementation is shown in Fig 2.13(a). It consists of a processor, data-memory and an instruction-memory, connected by a system bus. For our simulations, we use Intel’s programmable Nios-II processor [48], and extend the associated instruction set (ISA) to incorporate new custom-instructions enabled by our proposed X-SRAM (see Fig. 2.13(b)). The system bus follows the Avalon memory-mapped protocol, with enhanced architecture to enable passing three addresses at a time. Note that this is not a huge overhead since *in-memory* instructions do not pass the data operands, and thus the data-channel along with the address-channel can be used to pass three memory addresses over the bus. This methodology is similar to the work presented in [33]. A complete RTL model of the proposed X-SRAM was developed using the circuit parameters summarized in Table 1, incorporating the *in-memory* computation capabilities. We perform cycle-accurate RTL simulations to run the benchmark AES application [49] on the architecture described above. AES encryption algorithm heavily relies on substitution-and-permutation operations that utilize several bit-wise Boolean operations such as XORs, which makes X-SRAM custom

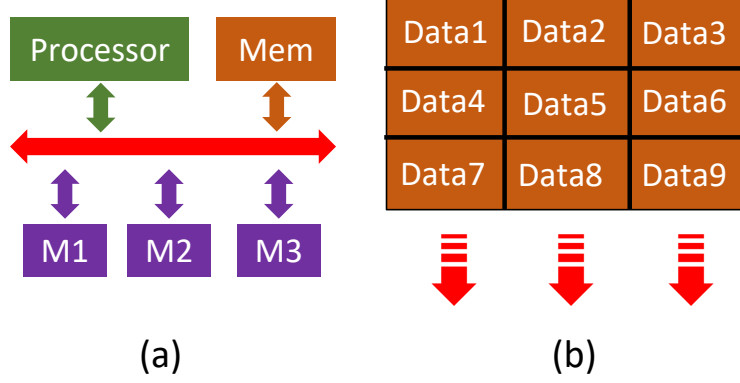


Figure 2.15. (a) Realistic scenario for a typical system with multiple masters over a shared bus. An arbiter keeps track of the memory traffic and controls which master has access to the bus at a given point in time. (b) Data-parallelism in memory arrays. X-SRAM performs bit-wise operations throughout the row, where each row may store multiple data words. Thus multiple computations occur in parallel.

instructions suitable for this application. We identified pieces of code constituting 92% of the entire runtime which can be mapped using the custom instructions RCS-XOR and RCS-Copy (shown in Fig. 2.13(b)), along with usual memory read-write instructions. The software was modified by replacing repetitive Boolean operations with our custom instruction macros.

2.5.2 Results and Discussion

We evaluate three modes of AES encryption and decryption namely CBC, CTR and ECB [50] for two different key sizes – 128bits and 256bits. We plot the total number of memory accesses (memory read instructions, memory write instructions and custom in-memory instructions) required for each mode in Fig. 2.14. The results are normalized to the corresponding memory accesses required in a conventional SRAM memory block (no *in-memory* custom instructions). The plots show that the memory accesses can be reduced by up-to 74.7% and 74.6% in ECB mode for 128b and 256b key respectively, by using X-SRAM in-memory instructions. The implications of these are threefold. 1) Since memory transactions are expensive, we directly save $\sim 75\%$ memory access energy consumption by reducing the number of accesses to the memory. The total energy consumption in the peripheral circuitry

is also thereby reduced. 2) In a realistic scenario, shown in Fig. 2.15(a), multiple masters access the shared system bus, thereby causing large arbitration delays. Reducing the total number of memory accesses allows the system bus to cater to other masters, thereby reducing arbitration wait times over the shared bus and hence improving overall system performance. 3) The decrease in the data transfer volume between the processor and memory alleviates the problems associated with limited bus bandwidth while providing enough memory bandwidth for parallelism. Fig. 2.15(b) shows how data can be mapped to the X-SRAM to exploit data-parallelism. Since X-SRAM has capability to compute two physical rows at a time, Data1-4, Data2-5 and Data3-6 can be computed in parallel with a single in-memory instruction, thereby improving throughput.

2.6 Conclusion

Von-Neumann machines have fueled the computing era for the past few decades. However, the recent emphasis on data intensive applications like artificial intelligence, image recognition, cryptography *etc.* requires novel computing paradigm in order to fulfill the energy and throughput requirements. ‘In-memory’ computing has been proposed as a promising approach that could sustain the throughput and energy requirements for future computing platforms. We have proposed multiple techniques to enable in memory computing in standard CMOS bit-cells – the 8T cell and the 8⁺T cell. We have shown that Boolean functions like NAND, NOR, IMP and XOR can be obtained by minimal changes in the peripherals circuits and the associated read-operation. Further, we have also proposed a ‘*read-compute-store*’ scheme by leveraging the decoupled read and write ports of the 8T and 8⁺T cells, wherein the computed logic data can be directly stored in the desired row of the memory array. Our results are supported by rigorous Monte-Carlo simulations performed using predictive transistor models. Moreover, taking an example of AES encryption algorithm, we demonstrate that up-to 75% memory transactions can be avoided, thereby allowing energy and performance improvements.

3. ACCELERATING BINARY CONVOLUTIONAL NEURAL NETWORKS IN 10T SRAM ARRAYS

3.1 Introduction

Deep convolutional neural networks (CNNs) have been established as the state-of-the-art for recognition and classification tasks [1], [2], often surpassing human capabilities [51]–[53]. Most popular networks that won the ImageNet [54] challenge, such as AlexNet [55], GoogLeNet [56], ResNet [52], *etc.*, are based on deep CNNs. However, hardware running these networks consume large amounts of energy, in fact, orders of magnitude more than the human brain [57]. This immense energy-gap stems from the underlying architecture of the current state-of-the-art hardware implementations, that are variants of the von-Neumann machines [5]. They contain physically separate computation and memory blocks, connected via a system bus. Although this architecture has worked wonders for general-purpose computing tasks, when it comes to deep CNNs and data intensive applications in general, frequent data transfers between the memory and the computation unit becomes a bottleneck, given the limited bandwidth of the bus. Moreover, since each transaction is expensive, a large power penalty is incurred per memory access.

Recent developments in the neural network community have identified these problems and have come up with simpler *memory-friendly* algorithms. Binary neural networks [6], [58], [59] and XNOR-nets [60] have been recently developed and shown large potential. The idea is to reduce the precision of input activations and the network weights to single-bit. This immensely simplifies the computations to Boolean bit-wise operations, with only minimal degradation in the state-of-the-art accuracies. Since convolution is the most power-hungry operation in neural networks, it is reduced to a bit-wise XNOR followed by a population count (*popcount*) of the XNORed output. This opens pathways for adopting new simplified binary *in-memory* computing paradigms for accelerating neural networks. As shown in [17], [61], [62], bit-wise Boolean operations including XORs or XNORs as well as non-Boolean vector-matrix dot-products can easily be incorporated within standard SRAM arrays. Such SRAM based *in-memory* computations open up new possibilities of augmenting the existing memory arrays with compute capabilities. Thereby, one can imagine a modified von-

Neumann machine, which can cater well to general purpose computing tasks as well as act as on-demand compute accelerator.

To that effect, we propose novel techniques to compute in-memory binary convolutions, as an added functionality to the standard 10-transistor (T) SRAM bitcells. In the first approach (Proposal-A), we use charge-sharing between the parasitic capacitances present inherently in the SRAM array to perform the XNOR and *popcount* operations involved in the binary convolution. Although this approach is digital, with binary weights and binary inputs stored in the memory array, the *popcount* is generated as an analog voltage on the source-lines. In order to sense this analog voltage, we propose a low-overhead and low-precision ADC (owing to area and energy constraints in the memory array). Another key highlight of this approach is that we employ a *sectioned-SRAM* by dividing memory sub-banks into smaller sections. With n -sections in a particular sub-bank we can accomplish n -binary convolutions in parallel. This is important because obtaining the *popcount* output for large kernels is non-trivial. For large networks, the kernel sizes in deeper layers are typically too large to be stored in a single row of a given memory sub-array. As such, *popcount* for larger binary networks inevitably requires a scheme to estimate the partial *popcount* from each row, which can then be summed up from different sub-arrays to get the final *popcount*. However, the low-overhead and low-precision ADC induces approximations in the *popcount* output, which results in overall system accuracy degradation. Thus, we propose another approach (Proposal-B), where we alter the peripheral circuitry of the SRAM array and enable two word-lines simultaneously. This approach, although not as energy/throughput efficient as Proposal-A, generates accurate XNOR and *popcount* operations, thereby, not affecting the overall system accuracy. The proposed circuit techniques in Proposal-A and Proposal-B allows us to process multiple kernels at once, thereby improving the overall system throughput and making the proposal suitable for a range of deep binary networks.

There have been several previous works to develop hardware platforms that can accelerate CNN algorithms. Hardware architectures that use highly sub-banked memory units feeding an array of multiply-accumulate processing elements have been presented in many works including [63]–[65]. A key drawback of such distributed processing array based customized design is the fact that it makes the underlying computing hardware application specific and

in many cases specific to neural network accelerators. Other works as in [66]–[68] have targeted in-memory charge-based computations in SRAM arrays. These computations can be applied to the case of accelerating binary neural networks by adding appropriate peripheral circuits for enabling binary convolutions. As compared to above works, we focus on the case of binary convolutions that allows us to achieve improved parallelism, geared specifically towards accelerating binary networks. Further, emerging technologies like memristive crossbars have been employed in many proposals as convolution accelerators for neural networks [32], [69]. The very use of memristors as convolution engine renders such platforms unsuitable for general purpose computing due to various challenges faced by memristive state-of-the-art technologies. These include, the limited endurance of memristive devices, the multi-cycle write-verify programming scheme [70] and the drift in programmed resistance state with aging [71]. More recently, [72] demonstrated an analog approach to binary convolution using charge-sharing in SRAM cells. However, the work presented in [72] was limited to smaller networks. This is because with larger and more complex networks, the inaccuracies in interfacing conventional low precision DACs/ADCs unacceptably degrades the network accuracy. Additionally, the work in [73], need a 12 transistor bit-cell with current-mode computations for binary convolutions and use of flash ADC with non-linear quantization.

In contrast to previous works, we show that using 10T SRAM cells in conjunction with few additional circuit techniques, fairly accurate binary convolutions can be performed with low-precision, low-overhead ADCs [74]. The main highlights are as follows:

1. We present two novel techniques to compute binary convolutions. Proposal-A uses charge-sharing between the parasitic capacitances inherently present within the standard 10T-SRAM array, to accomplish a fairly accurate *popcount* operation. Proposal-B alters the SRAM peripheral circuitry to perform accurate in-memory XNOR and *popcount* operations.
2. Further, we propose *sectioned-SRAM* to increase parallelism within the SRAM arrays, thereby improving the computation throughput and energy-efficiency of the binary convolution operation.

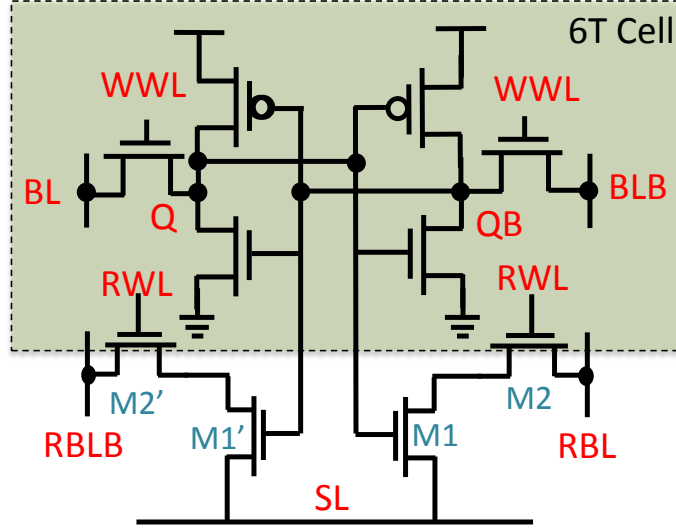


Figure 3.1. The 10 transistor SRAM cell featuring a differential decoupled read-port comprising of transistors M1-M2 and M1'-M2'. The write port is constituted by write access transistors connected to WWL.

3.2 In-memory Binary Convolution – Proposal-A

As discussed in the introduction, a convolution operation is simplified to a bitwise XNOR, followed by a *popcount* of the XNORed output in binary neural networks (BNNs). Although bitwise XNOR operation is simple to incorporate within the memory, the *popcount* operation is not very straightforward. We exploit the inherent SRAM structure, utilizing the internal parasitic capacitances to perform the XNOR and *popcount* of two vectors stored within the memory array. Although our approach to binary convolution is digital, we sense an analog voltage within the memory array to evaluate the *popcount* output. Sensing analog voltages in general, is difficult without precise ADCs. Most common precise ADCs, such as Flash ADCs and SAR type ADCs require excessively large power and area [75], making them unsuitable for memory applications. Thus, we propose a dual read-wordline (Dual-RWL) along with a dual-stage ADC to minimize the errors in the *popcount* output. Further, we describe the sectioned-SRAM technique to improve the throughput and the energy-efficiency of the binary convolution.

3.2.1 Circuit Description

We use a standard 10T-SRAM cell as the basic memory unit. Fig. 3.1 shows a schematic of the 10T-SRAM cell, containing the basic 6T-cell as the storage unit, along with transistors M1-M2 and M1'-M2' forming the differential read ports, respectively. Writing into the cell is functionally similar to the 6T write operation through the write-ports (WWL, BL, BLB). For reading, RBL and RBLB are pre-charged to V_{DD} , SL is connected to ground, and RWL is enabled. If the bit-cell stores a '1' ($Q = V_{DD}$, $QB = 0V$), RBL discharges to 0V and RBLB holds its charge. Similarly, if the bit-cell stores a '0' ($Q = 0V$, $QB = V_{DD}$), RBLB discharges to 0V and RBL holds its charge. A differential sense amplifier senses the voltage difference between RBL and RBLB to generate the output. As will be apparent in the following sub-sections, the choice of 10T-cell as opposed to 6T- or 8T- cell was based on two factors 1) the 10T-cells allow us to implement in-situ XNOR operation through the transistors M1-M2 and M1'-M2' 2) it also helps to accumulate a resultant voltage on the SL line which is proportional to the overall pop-count. Thus, the availability of the differential decoupled read-port in the 10T-cell can be used to achieve both the XNOR computation and estimation of the pop-count operation.

We use the inherent parasitic capacitances on RBLs, RBLBs and SLs (C_{RBL} , C_{RBLB} and C_{SL} , respectively) in the 10T-SRAM structure to compute the binary convolution within the memory array itself. The operation can be described in three steps as follows:

Pseudo-read: A read operation is performed on a row storing the binary vector inputs, say A1 (refer Fig. 3.2(a)). First, all RBLs/RBLBs are precharged to V_{DD} , as in the usual read operation. Next, when the RWL corresponding to the row storing A1 is enabled, the precharged RBLs and RBLBs discharge conditionally, depending on the data values, thereby stabilizing at V_{DD} or 0V. For the example shown in the figure, the data stored is '1' in both cells corresponding to the input vector A1, thus, both RBLs discharge to 0V and RBLBs stay at V_{DD} . Note that the differential sense-amplifiers are not enabled in this *pseudo-read* step.

XNOR on SL After the pseudo-read operation, the RBLs/RBLBs store the information of A1 as their respective voltages. Now, the RWL of the row storing a weight kernel, say

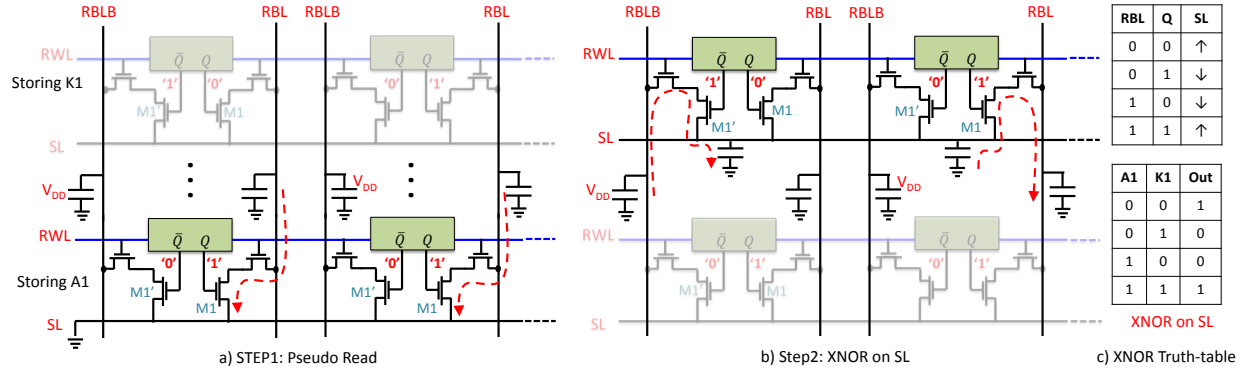


Figure 3.2. Illustration of the binary convolution operation within the 10T-SRAM array. a) Step 1: Pseudo-read. RBLs/RBLBs are pre-charged and RWL for a row storing the input activation (A1) is enabled. Depending on the data A1, RBLs/RBLBs either discharge or stay pre-charged. The SAs are not enabled, in contrast to a usual memory read. Thus, the charge on RBLs/RBLBs represent the data A1. b) Step 2: XNOR on SL. Once the charges on RBLs/RBLBs have settled, RWL for the row storing the kernel (K1) is enabled. Charge sharing occurs between the RBLs/RBLBs and the SL, depending on the data K1. The RBLs either deposit charge on the SL, or take away charge from SL. c) The truth table for Step 2 is shown. The pull-up and pull-down of the SL follow the XNOR truth table. Moreover, since the SL is common along the row, the pull-ups and pull-downs are cumulative. Thus, the final voltage on SL represents the XNOR + *popcount* of A1 and K1.

K1, is enabled (refer Fig. 3.2(b)). Interestingly, this causes charge-sharing between C_{RBL} , C_{RBLB} and C_{SL} as shown in the figure by the charge current paths. In the example, the two cells corresponding to K1 store a '0' and '1' respectively. Thus, when the RWL is enabled, charge flows into the SL from M1' in the left cell, while charge flows out of SL through M1 in the right cell. This 'pull-up' (↑) and 'pull-down' (↓) of the SL follows the XNOR operation of the data stored in the cell (K1) and the RBL/RBLB charge (A1). With respect to the example chosen above, one can observe that the first two rows of the XNOR truth table of Fig. 3.2(c) are taken care-of. If the bits corresponding to the activation (A1) was '0' and '0', *i.e.*, RBLB is at 0V while RBL is at V_{DD} , then the charge flows out of SL through M1' in left cell, while it flows into SL through M1 in right cell. This represents the bottom two rows of the XNOR truth table. Thus, we perform a bitwise XNOR operation between vectors A1 and K1, represented by the charge stored on the line SL.

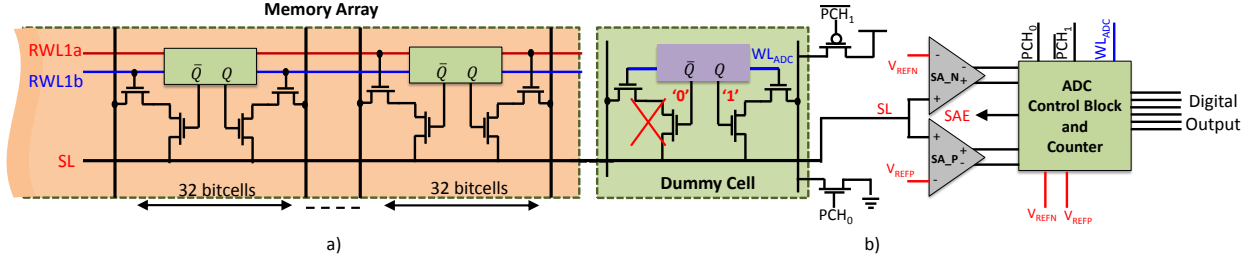


Figure 3.3. a) Dual RWL technique. b) Dual-stage ADC scheme.

Popcount Since the SL is shared by all the cells along the row, these ‘pull-ups’ and ‘pull-downs’ are cumulative. As can be seen from Fig. 3.2(c), an SL ‘pull-up’ corresponds to a ‘1’ in the output XNORed vector, while an SL ‘pull-down’ corresponds to a ‘0’ in the output XNORed vector. In order to evaluate the *popcount* of the output vector, we need to count the number of 1’s. More 1’s in the output vector implies more ‘pull-ups’ on SL, which in turn implies a higher voltage on SL. Thus, the final SL voltage represents the *popcount* of the output vector: (A1 XNOR K1). We boost the RWL voltage such that the SL swing is from 0V to V_{DD} . To sense this analog voltage we use a charge-sharing based sequentially integrating ADC, adopted from [72]. Note however, that this is an approximate, low-precision ADC. Thus, in order to achieve a fairly accurate estimate of the *popcount* of the entire row, we use two techniques described in the next sub-section.

3.2.2 Dual Read-Wordline based Dual-stage ADC

In order to evaluate the *popcount* of the entire memory row at once, we should be able to distinguish N- number of distinct states in the analog SL voltage, where N is the number of columns in a memory array (we choose N=64, for a reasonably sized array). In the output XNORed vector, there can zero 1’s, one 1’s, two 1’s, ... , up to N 1’s. Correspondingly, there are N- different voltage levels on the SL, which need to be sensed by the ADCs. However, due to area and power constraints within the memory array, it is infeasible to use high-precision ADCs, such as the area-expensive SAR or power-hungry Flash ADCs. We adopt a simple charge-sharing based serially integrating type ADC for our purposes. However, instead of

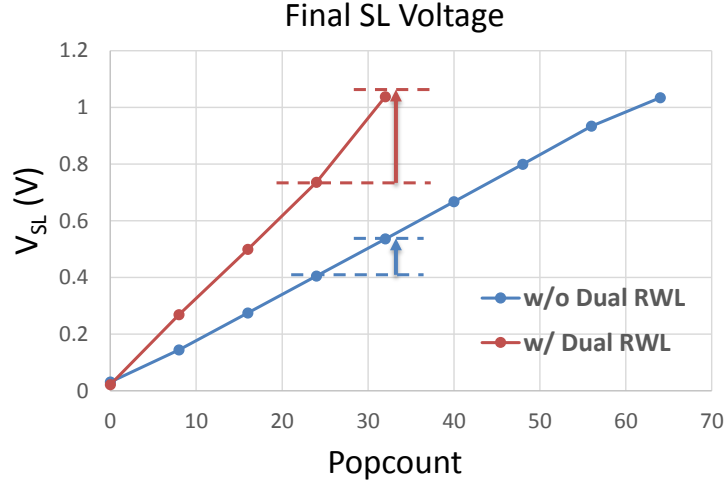


Figure 3.4. The plot shows the final SL voltage with and without the Dual RWL approach. A larger sense margin is obtained with our Dual RWL approach, thus relaxing the constraints on the low-overhead ADC. Note that with Dual RWL technique we restrict the distinct voltage levels on SL to 32 at a time, instead of 64. However, the voltage swing on SL remains the same, thereby increasing the sense margin between the states.

having to sense N - distinct analog levels, the ADC only needs to sense $N/8$ levels. This is enabled by using a Dual RWL memory structure, along with a dual-stage ADC.

The Dual RWL technique is shown in Fig. 3.3(a). Note that we use two sets of read word-lines (RWL1a, RWL1b) for every memory row. First half of the cells along the row are connected to RWL1a, while the rest are connected to RWL1b. The step 2 of the binary convolution (XNOR on SL) described above is split in two parts. First, only RWL1a is enabled. Thus, only $N/2$ cells are enabled to share charge with SL, either pulling-up or pulling-down the SL voltage. The rest half of the cells are cut off from the SLs, and cannot participate in the charge sharing. Once the SL voltage has been sensed by the ADC, RWL1a is disabled, and RWL1b is enabled. Now, the other half of the cells share charge to generate a voltage on SL. Note that this does not change the swing on the SL, since the SL voltage depends on the capacitive ratio C_{RBL}/C_{SL} . Thus, the $N/2$ voltage levels are equally separated out from 0V to V_{DD} . This can be confirmed from Fig. 3.4, which shows the SL voltages for $N=64$, with and without Dual RWL technique, as a function of the *popcount*. Since the separation

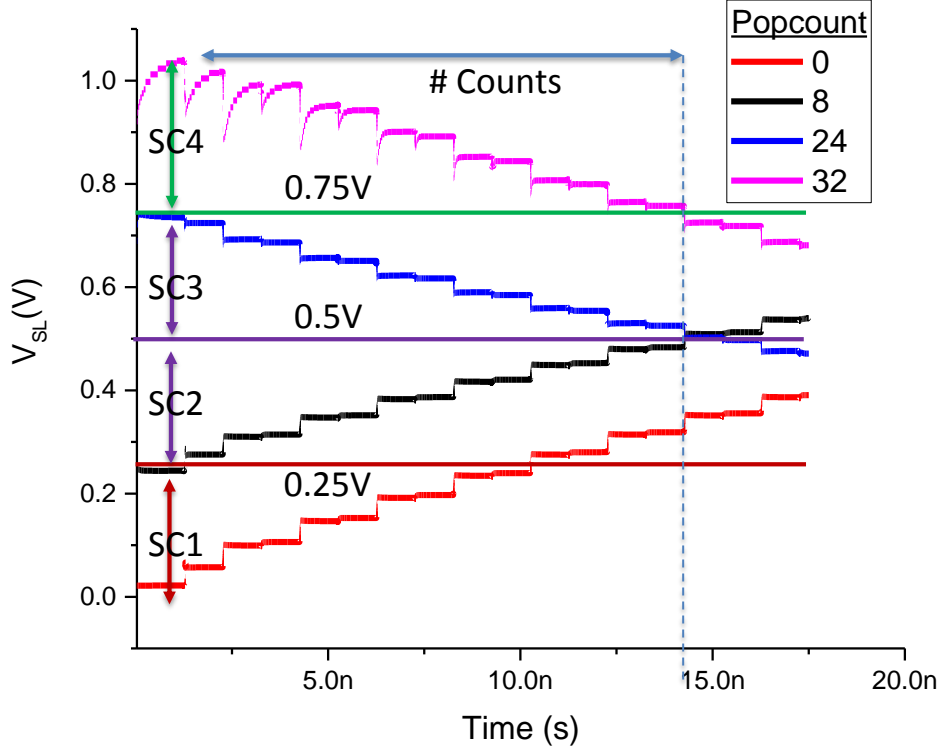


Figure 3.5. The figure shows the timing diagrams for the dual-stage ADC scheme. The figure plots the SL voltage for various *popcount* cases. In the first-stage, the sub-class SC1-4 is determined using multiple references (0.25V, 0.5V and 0.75V). In the second-stage, charge is pumped-in/out of SL successively, depending on the SC. The number of cycles it takes for SL to reach V_{REF} are counted. V_{REF} for SC1-4 is 0.25V, 0.5V, 0.5V and 0.75V, respectively.

between the states has increased, it becomes easier to sense the levels with a low-overhead ADC.

The ADC used is shown schematically in Fig. 3.3(b). It consists of two dummy bitcells per row (only 1 shown in figure), two SAs, a counter and an ADC logic block. We employ a dual-stage ADC to sense the analog voltage on SL. In the first-stage for ADC sensing, we use multiple voltage references ($V_{DD}/4$, $V_{DD}/2$ and $3V_{DD}/4$), to classify the analog voltage levels into four sub-classes SC1, SC2, SC3, SC4 – $[0-V_{DD}/4]$, $[V_{DD}/4-V_{DD}/2]$, $[V_{DD}/2-3V_{DD}/4]$ and $[3V_{DD}/4-V_{DD}]$, respectively. This is done using two voltage SAs, since the voltage swing on SL spans 0V to V_{DD} . On SA_N, a V_{REF} of $3V_{DD}/4$ is applied, while for SA_P, a V_{REF} of

$V_{DD}/4$ is applied. If both SA outputs are LOW, the SL voltage is classified in SC1. Similarly, when both SA outputs are HIGH, the SL voltage is classified in SC4. Otherwise, V_{REF} is changed to $V_{DD}/2$, and the SA outputs are observed again. If both outputs are HIGH, the SL voltage is classified in SC3, otherwise SC2. Thus, the first-stage of the ADC generates the MSB 2bits of the ADC output.

Once the sub-classes of the analog voltage have been defined, the second-stage of the ADC is initiated. The ADC logic block generates a set of control signals – PCH_0 , $\overline{PCH_1}$ and WL_{ADC} , which operate on the dummy bitcells. For SC1 and SC2, SA_P is enabled with a V_{REF} of $V_{DD}/4$ and $V_{DD}/2$, respectively. $\overline{PCH_1}$ is pulsed alternately with WL_{ADC} , to pump-in a small amount of charge into SL every cycle through the dummy cells. In each cycle, when WL_{ADC} is LOW and $\overline{PCH_1}$ is HIGH, the RBL of the dummy cell is precharged to V_{DD} . When WL_{ADC} is HIGH and $\overline{PCH_1}$ is LOW, the precharged RBL pumps-in charge to the SL. In successive cycles, the voltage on SL increases. As soon as the SL voltage exceeds V_{REF} , SA_P output flips from LOW to HIGH. The number of cycles in the process are counted using a digital counter. On the other hand, for sub-classes SC3 and SC4, SA_N is enabled with a V_{REF} of $V_{DD}/2$ and $3V_{DD}/4$, respectively. PCH_0 is enabled instead of $\overline{PCH_1}$, thereby pumping-out charge from SL every cycle. Again, the number of cycles are counted when SA_N flips from HIGH to LOW. This is illustrated in Fig. 3.5, which shows the operation of ADC taking an example of *popcount* cases 0, 8, 24 and 32, for N=64. For the *popcount* case 32 and 24, the sub-classes SC4 and SC3 are determined respectively, thus, charge is pumped out of SL every cycle. Similarly for the *popcount* cases 8 and 0, SC2 and SC1 are determined respectively, and charge is pumped into SL every cycle. The two dummy bitcells are used to mimic the capacitances of the RBLs/RBLBs, such that the charge being pumped in/out from SL every cycle by the dummy bitcells mimics the charge sharing of RBLs/RBLBs and SL in Step 2 (XNOR on SL) operation. Note that the amount of charge being pumped-in/pumped-out exponentially decreases with time. This is a fundamental limit to charge-sharing type ADCs and thus, they work only if the number of counts are small. In our case, for N=64, we count only $N/8 = 8$ states using this ADC, which gives us fairly accurate results, as shown later. Thus, the output from the first-stage (sub-class SC1-4) along with the output from the second-stage (ADC counts) estimates the number of

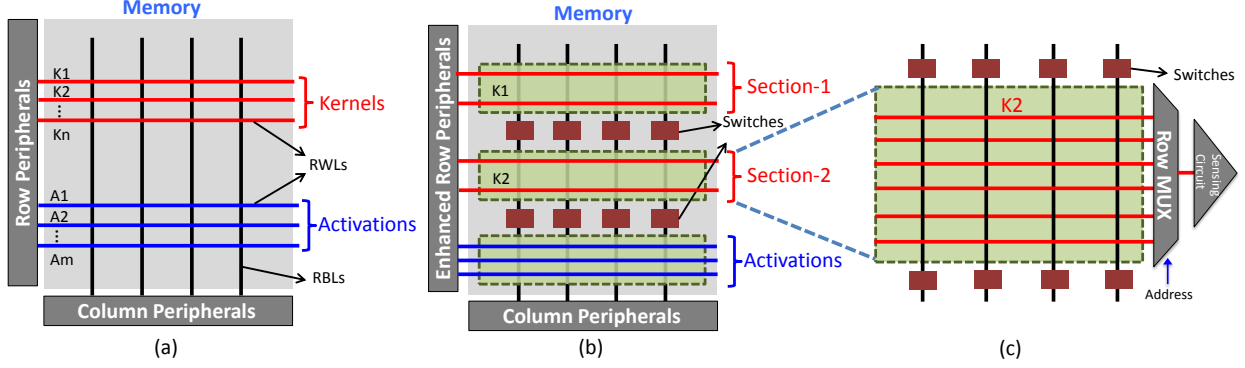


Figure 3.6. a) Typical SRAM memory array with row and column peripherals storing the activations $A1-A_m$, and kernels $K1-K_n$. b) Proposed sectioned-SRAM array. By introducing switches along the RBLs, the array is divided into sections. The kernels are mapped into the sectioned-SRAM with each section storing different kernel. Once the activations are read onto the RBLs, the switches are opened, and the memory array is divided into sections. c) Since the RBLs for each section have been decoupled, one RWL in each section can be simultaneously enabled such that each section performs the binary convolution concurrently. The Row-MUX connects the corresponding SL to the sensing circuit. For example, if $A1$ was read onto the RBLs before sectioning, enabling the rows $K1$ and $K2$ in Section 1 and 2 respectively, we obtain $A1 \cdot K1$ and $A1 \cdot K2$ in parallel on the SLs, which are sensed by the ADC.

1's (*popcount*) for the XNORed output vector. Note that two sets of *popcounts*, one from RWL1a and other from RWL1b, are sequentially read, and then added together to get the final *popcount* of the vector.

3.2.3 Sectioned Memory Array for Parallel Computing

We have seen that XNOR and *popcount* operations can be computed within the SRAM array. The manner in which these computations are done, opens possibilities for improving the throughput and energy-efficiency in performing binary convolutions. A typical operation in a CNN layer involves *convolution* of input activations with multiple kernels. This gives us an opportunity for data re-use, since the same set of activations need to be convolved with different kernels. Our proposed scheme described above is well suited to exploit this property of CNNs. Given a set of activations $A1, A2, \dots, A_m$ and kernels $K1, K2, \dots, K_n$, stored within the memory array (see Fig. 3.6(a)), we need to compute $A1 \cdot K1, A1 \cdot K2, \dots, A1 \cdot K_n$,

$A2 \cdot K1, A2 \cdot K2, \dots, A2 \cdot Kn$ and so on. In our computations described above, specifically in the *pseudo-read* step, the data corresponding to $A1$ is read onto the RBL/RBLB voltages. We propose sectioning the memory array into subsections by introducing switches along the RBLs, as shown in Fig. 3.6(b), such that kernels are grouped into different sections. Each section consists of a separate row-multiplexer and ADC control block, as shown in Fig. 3.6(c). The row multiplexer connects the selected SL to the ADC. After $A1$ has been read onto the RBLs/RBLBs, the switches are opened. The RBLs/RBLBs in individual sections store the information of data $A1$, but have been decoupled. This allows us to enable one memory row in all the sections corresponding to kernel $K1$ in section 1, $K2$ in section 2, and so on, thereby evaluating the XNOR-*popcount* operations concurrently, in all n sections. We thus obtain the output $A1 \cdot K1, A1 \cdot K2, \dots, A1 \cdot Kn$ in a single cycle. This step can be repeated for all activations $A1, A2, \dots, Am$. Thus, sectioning the memory array improves the throughput of our computations n -fold. Moreover, with a single pseudo-read step, we are able to perform n convolutions, thereby saving multiple pseudo-read cycles which consume bitline precharge energy. Specifically, without sectioning, one RBL and RBLB pre-charge is required for every convolution operation in addition to ADC energy consumption. With n -sections per sub-bank we obtain n -convolutions per pre-charging of the RBL and RBLB thereby not only increasing parallelism but also energy-efficiency. Note that the number of sections (n) is carefully chosen based on the SRAM array design. When the sectioning switches are opened, the effective capacitance of the RBLs/RBLBs in each section is reduced, thereby reducing their charge sharing capability. Subsequently, the maximum voltage swing obtained on the SL is affected. This limits the maximum size of each section that can be introduced into the array. One also has to take into consideration that the SL capacitance is charged or discharged by multiple BLs (or BLBs) that effectively come in parallel and increase the overall charging/discharging swing of the SL line. Taking into consideration the aforementioned factors, we sectioned a $128 \times 64b$ array into 4 sections, each of size $32 \times 64b$.

Let us now discuss how binary convolutions can be obtained for large kernels using the distributive property of *popcount*. Note that our memory array size is not related to the kernel size for a particular network. If the kernel size is larger than the memory word length, which is often the case in deeper state-of-the art CNN layers, a single kernel occupies multiple

rows in the same or different sub-banks. In-memory binary convolution is performed for each of these kernel rows separately, and the partial *popcounts* obtained from each operation are added to generate the final *popcount*.

$$\text{popcount}(N + M + \dots) = \text{popcount}(N) + \text{popcount}(M) + \dots$$

Once the final *popcount* is obtained, the output of the binary convolution operation is ‘1’ if the final *popcount* (number of 1’s) is greater than half the kernel size, and ‘0’ otherwise. This allows our design to be generic, and scalable to deeper networks.

3.2.4 Results

The sectioned-SRAM array assuming a section size of 32 rows and 64 columns was simulated in HSPICE using the 45-nm predictive transistor models (PTM) [46]. As described in the previous section, the final voltage at SL denotes the *popcount* output of the binary convolution. The SL voltage is sensed using the ADC described in the previous section. Again, the 45-nm PTM models were used to simulate the SA and the ADC logic block. Using the Dual RWL along with a dual-stage ADC, the ADC output is relaxed to only 5bits. The most-significant bits (2bits) are generated in the first-stage of the ADC (sub-classes SC1-4) using multiple references, while the lower bits (3bits) are generated in the second-stage by the integrating ADC. We observe the effects of CMOS process variation on the ADC output using Monte Carlo simulations, in presence of 30mV sigma threshold voltage variation. Fig. 3.7 plots the distribution of the second-stage ADC output for various *popcount* cases. Note that a similar trend repeats for higher *popcount* cases with modulo-8, since only the lower 3bits of the output are generated in the second-stage. The ADC output is fairly accurate with a small overlap with the neighboring counts. The small inaccuracy is attributed to the transistor threshold voltage variations in the memory array and in the SAs used in the ADC. Moreover, the charge being pumped in/out of SL decreases with each cycle, due to charge-sharing, thereby inducing errors for higher counts. The inset shows a best-fitting normal distribution for the variations in the ADC output. The average standard deviation of the counts was found to be ~ 0.4359 counts. Total energy consumed per operation was

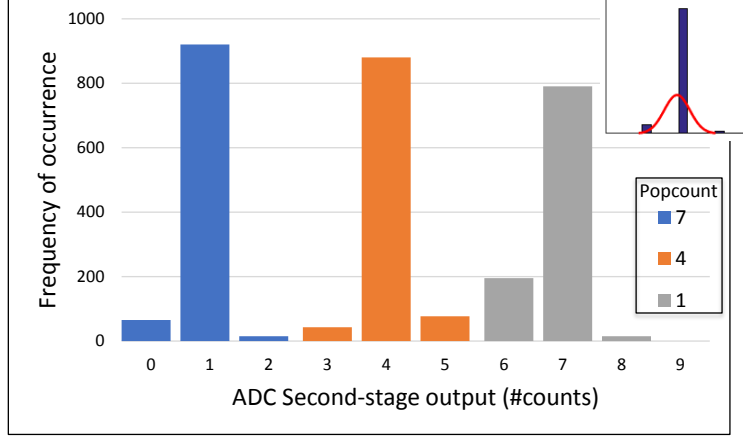


Figure 3.7. Monte-Carlo simulations. The figure plots the histogram of the second-stage output of the ADC, for various *popcount* cases, in presence of process variations. Inset: Each histogram is fitted with a Gaussian distribution. The average standard deviation of the counts is ~ 0.4359 . The trend repeats for higher *popcount* cases with modulo-8, since only the lower 3bits of the output are generated in the second-stage.

estimated to be $\sim 0.767\text{pJ}$ and $\sim 1.914\text{pJ}$, with and without sectioned-SRAM (4 sections per bank), respectively. The energy was averaged over various *popcount* cases. Here, by one operation, we mean XNOR + *popcount* of a 64bit input activation and a 64bit kernel, both of which are stored in the SRAM. The energy consumption includes the pre-charge energy in the pseudo-read step and the ADC energy. The latency of one operation was $\sim 45\text{ns}$. This is due to the low-overhead integrating ADC used, which serially counts to estimate the *popcount*.

3.3 In-memory Binary Convolution – Proposal-B

In the previous section, we described an energy-efficient implementation of performing binary convolutions within the SRAM array. However, the low-overhead ADC used to determine the *popcount* induces errors in the convolution output, which may impact the system accuracy, as we will show later. The primary cause of the inaccuracy is the generation and detection of an analog voltage, which is susceptible to noise, offset etc. Thus, in this section, we propose yet another implementation of enabling binary convolutions in standard SRAM arrays, by modifying the peripheral circuitry. This approach is robust since the *pop-*

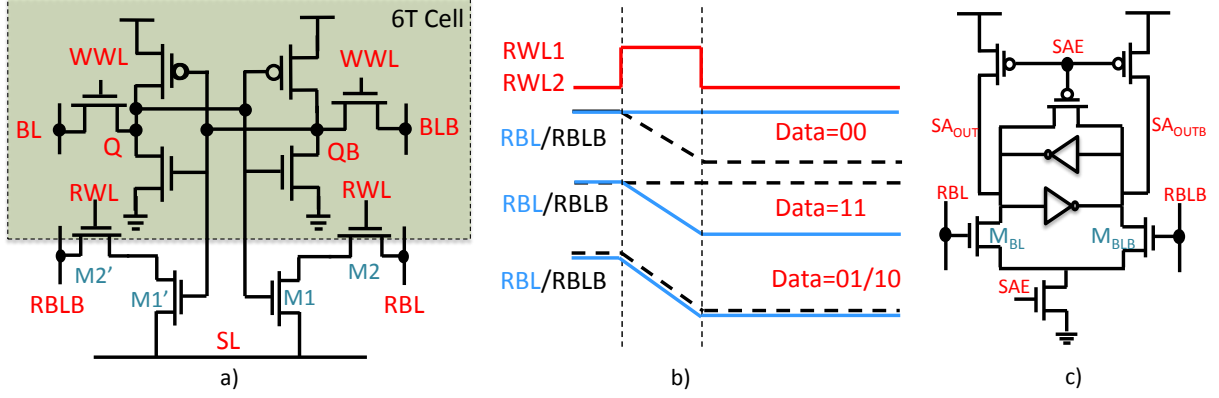


Figure 3.8. (a) A 10T-SRAM bitcell schematic is repeated here for convenience. (b) Timing diagram used for in-memory computing with 10T-SRAM bitcells. (c) Circuit schematic of the asymmetric differential sense amplifier. [61]

count is computed using digital logic gates (full-adders), unlike Proposal-A which uses analog voltages. Although this robustness comes at a cost of energy-efficiency and throughput as compared to the previous proposal based on charge-sharing, our simulations show that this implementation is still better than the typical von-Neumann based approach as it leverages *in-memory computing* for XNORs and pop-count operations.

3.3.1 Bitwise XNORs

Bitwise Boolean operations within SRAM arrays have recently been demonstrated in [11], [17], [61]. The idea is to enable two RWLs together during a read operation. Let us consider words ‘A’ and ‘B’ stored in two rows of the memory array. Note that we can simultaneously enable the two corresponding RWLs without worrying about read disturbs, since the bit-cell has decoupled read-write paths (shown in Fig. 3.8(a)). The RBL/RBLB are pre-charged to V_{DD} . For the case ‘AB’ = ‘00’ (‘11’), RBL (RBLB) discharges to 0V, but RBLB (RBL) remains in the precharged state. However, for cases ‘10’ and ‘01’, both RBL and RBLB discharge simultaneously. The four cases are summarized in Fig. 3.8(b). Now, in order to sense bit-wise XNOR from the RBL/RBLB voltages, we use two asymmetric SAs (see Fig. 3.8(c)[61]) which compute the bitwise NAND/NOR in parallel. Asymmetric SAs work by

sizing either one of the transistors M_{BL}/M_{BLB} bigger than the other. In Fig. 3.8(c), if the transistor M_{BL} is sized bigger compared to M_{BLB} , its current carrying capability increases. Thus, for cases ‘01’ and ‘10’ where both RBL and RBLB discharge simultaneously, SA_{out} node discharges faster, and the cross-coupled inverter pair of the SA stabilizes with $SA_{out}='0'$. While for the case ‘11’(‘00’), RBL(RBLB) starts to discharge, and RBLB(RBL) is at V_{DD} , making $SA_{out}='1'$ (‘0’). Thus it can be observed that SA_{out} generates an AND gate (thus, SA_{outb} outputs NAND gate). Thus, we call this sense-amp SA_{NAND} . Similarly, by sizing the M_{BLB} bigger than M_{BL} , OR/NOR gates can be obtained and we call it SA_{NOR} . Next, by ORing the NOR and AND outputs obtained from SA_{NOR} and SA_{NAND} respectively, bitwise XNOR operation is realized. A detailed description of the bit-wise Boolean XNOR can be found in [61].

3.3.2 Popcount

In order to utilize the above mentioned approach for enabling binary convolutions, we propose to add a *bit-tree adder* after the asymmetric-SA stage to generate the *popcount*, as shown in Fig. 3.9. The bit-tree adder consists of multiple full-adder (FA) blocks connected in a tree manner. The bit-tree adder sums up all the bits of the output XNORed vector to generate the *popcount*. The first layer of the bit-tree adder consists of single FA blocks, each of which is capable of adding three consecutive bits to generate a 2-bit output. In the next layer, 2-bit adders are used, which are constructed using two stacked FA blocks. The second layer generates 3-bit output. In subsequent layers, multiple FA blocks are stacked to construct multi-bit adders. Finally in the $\log(N)$ layer, where N is the number of columns in the sub-array, the *popcount* output is generated, and is read out from the memory. By enabling RWLs corresponding to rows storing activation (A1) and kernel (K1), the asymmetric-SAs generate the XNORed vector. The output XNORed vector is passed to the bit-tree adder, to generate the *popcount*.

To incorporate convolutions with large kernel sizes, the partial *popcount* generated from the bit-tree adders can be summed up over multiple cycles, to generate the final *popcount*. Note that the generated *popcount* is exact, as it is computed using conventional digital logic

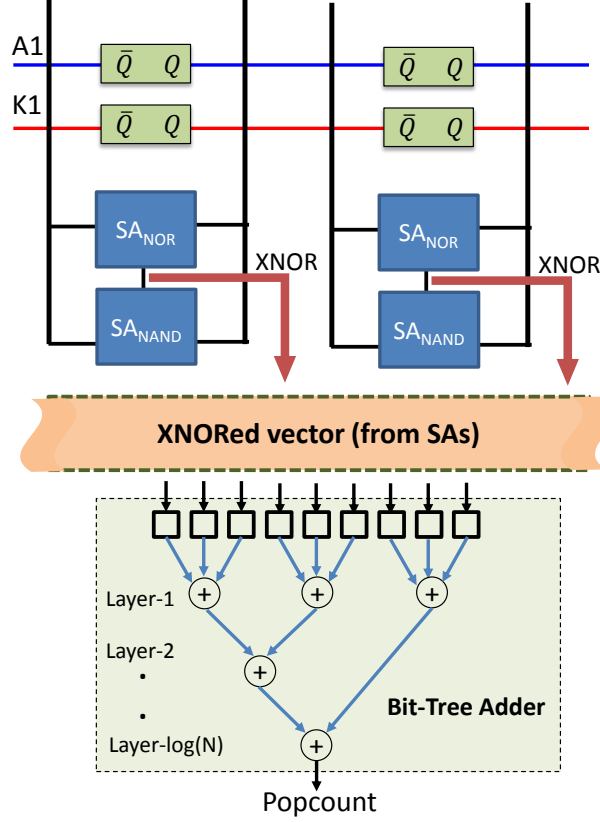


Figure 3.9. Modified peripheral circuitry of the SRAM array to enable binary convolution operations. It consists of two asymmetric SAs - SA_{NOR} and SA_{NAND} which pass the XNORed data vector to a bit-tree adder. The adder has $\log(N)$ layers, where N is the number of inputs to the adder. It sums the input bits to generate the *popcount*.

gates. Also note that the sectioned-SRAM concept described in the previous section is not applicable for this proposal.

3.3.3 Results

A 128×64 -bit SRAM array along with the asymmetric SAs - SA_{NOR} and SA_{NAND} were simulated in HSPICE using the 45-nm predictive transistor models (PTM) [46]. As described above, two RWLs are enabled simultaneously, and depending on the data stored in each of the bits, SA_{NOR} and SA_{NAND} generate bitwise NOR/OR and NAND/AND, respectively. Readers are referred to [61] for more circuit details and simulations. The energy

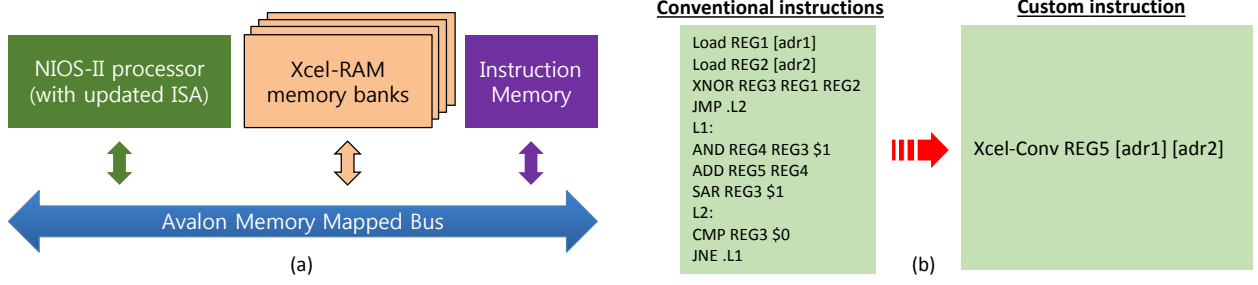


Figure 3.10. (a) Modified von-Neumann architecture based on Xcel-RAM memory banks and enhanced instruction set architecture (ISA) of the processor. (b) Snippet of assembly code for performing a binary convolution operation using conventional instructions and custom instructions.

consumption and latency of the bitwise XNOR operation was estimated to be 29.67fJ/bit and 1ns, respectively. The energy consumption includes the pre-charge energy and the energy consumed in asymmetric-SAs. The bit-tree adder was modeled in Verilog, and synthesized using Synopsys Design Compiler to the 45-nm tech node. The bit-tree adder is constructed using multiple FA (Full-Adder) blocks stacked in a tree fashion. The inputs to the bit-tree adder block are 64 wires, which represent the bitwise XNORed data generated from the SA stage. The output is a 6-bit *popcount*. The total area and power of the bit-tree adder in performing a 64-bit *popcount*, was estimated to be $523.5\mu m^2$ and 0.26mW, respectively.

3.4 System-level Evaluation Framework for BNN

In this section, we describe the framework developed to evaluate the benefits of our proposals at a system-level, taking an example of a deep binary neural network. We use a modified von-Neumann based system architecture, where the SRAM banks are replaced with our proposed Xcel-RAM banks (Proposal-A/Proposal-B) with embedded convolution compute capabilities. By utilizing these *in-memory* convolutions, we demonstrate the benefits in the overall system energy consumption and latency per inference.

3.4.1 Simulation Methodology

The modified von-Neumann processing architecture is shown in Fig. 3.10(a). It consists of a processor, an Xcel-RAM memory-block and an instruction-memory, connected by a system bus. The Xcel-RAM block consists of multiple subarrays that are arranged in a typical banked structure. We use the CACTI tool [76] to model a 64KB Xcel-RAM bank. The circuit specific energy and delay numbers obtained from HSPICE with the 45nm PTM models [46] were put in CACTI to obtain the per-access energy and latency of memory read/write operations as well as binary convolution operation. These include the energy consumed in H-trees, WL decoders, BL drivers, SAs, muxes etc. Next, a cycle-accurate RTL model was developed for Xcel-RAM banks, which was integrated with Intel’s programmable Nios-II processor [48], with instruction set (ISA) extensions to leverage the Xcel-RAM compute capabilities (see Fig. 3.10(b)). The system bus follows the Avalon memory-mapped protocol, with enhanced bus architecture to support passing multiple addresses at a time. Note that this is not a large overhead since *in-memory* instructions do not pass the data operands, and thus the data-channel is used to pass extra memory addresses over the bus [77]. Note that although we show a typical von-Neumann based system, Xcel-RAM banks can be interfaced with general purpose graphics processing units (GP-GPUs) based systems as well, to leverage data parallelism along with *in-memory computing*. Our aim here was to show the benefits of replacing conventional SRAM banks with compute capable Xcel-RAM banks.

The binary neural network (BNN) proposed in [6] uses binary bipolar activations (± 1) for both weights and activations. Note that in our memory, $+1$ is stored as logic HIGH bit, while -1 is stored as logic LOW bit. We used Pytorch platform [78] to train a BNN using the algorithm proposed in [6], taking help from their github repository [79]. The neural network architecture is given in Table 3.1. The network was evaluated on CIFAR-10 [80] and SVHN [81] datasets. All layers were binarized, except Conv1 and FC3 layers. It was observed that $\sim 99.4\%$ of total computations occur in the binarized layers - Conv2-6 and FC1-2, all of which can utilize the Xcel-RAM convolution capabilities (see Table 3.1). Or in other words, $\sim 99.4\%$ of total computations per-inference can be mapped using custom Xcel-RAM instructions, thereby giving us significant improvements in energy and throughput.

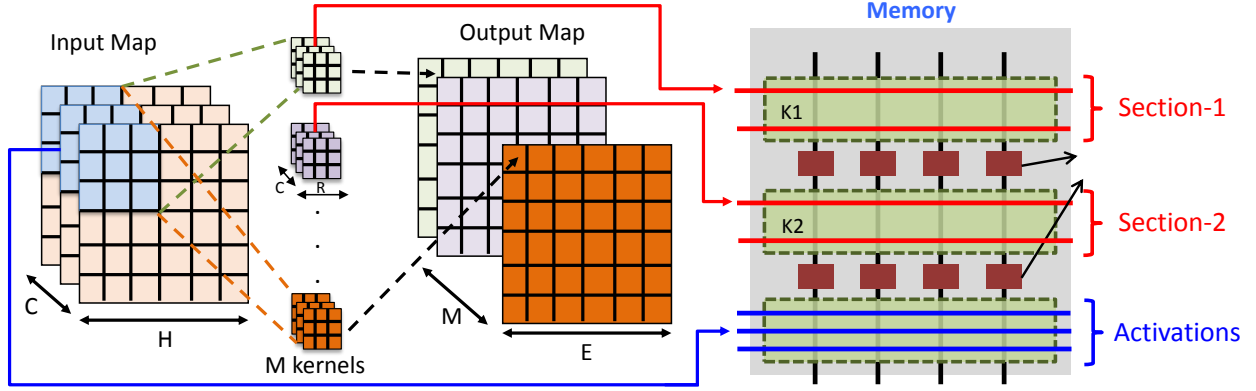


Figure 3.11. Mapping of weights and activations of a convolutional neural network to Xcel-RAM. The kernels and the input feature maps are flattened and stored into multiple rows in the memory array. Xcel-RAM banks have dedicated rows for storing kernels and activations.

Each of these layers were run on the modified von-Neumann architecture described above. We assume that the binarized kernels are stored in an off-chip memory, and the kernels for a particular layer are loaded into the SRAM before processing that layer. Typical values of DRAM access energy and latency were taken from literature [82]. The software was modified by replacing repetitive convolution operations with our custom instruction macros. In every layer, the convolutions are split into multiple 64-bit XNOR+*popcount* operations, which are then accumulated to compute the final output. The final output is stored back into the SRAM, which would be the input activations for the succeeding layer.

As a baseline, we use a similar system architecture, but with standard SRAM banks with only read/write capability, instead of Xcel-RAM banks. The convolution operation is performed in software through conventional instructions. A snippet of the assembly code for convolution in the baseline and Xcel-RAM based designs is shown in Fig. 3.10(b).

3.4.2 Mapping Weights and Activations to Xcel-RAM

Let us now discuss how the weights and activations of the BNN are mapped to Xcel-RAM banks, in order to fully utilize the in-memory compute capabilities. Fig. 3.11 shows one of the layers of binary convolutional neural network, having an input feature map of size $H \times H \times C$ and N kernels, each of size $R \times R \times C$. This results in, say, an output feature

Table 3.1. Benchmark Binary Neural Network [6] used for classifying CIFAR10 and SVHN datasets.

CIFAR10		SVHN	
Layer Description	% Computations	Layer Description	% Computations
128 3x3 Conv1	0.51	64 3x3 Conv1	0.88
128 3x3 Conv2	21.78	64 3x3 Conv2	18.84
[2 2] MaxPool1	0.04	[2 2] MaxPool1	0.07
256 3x3 Conv3	10.89	128 3x3 Conv3	9.42
256 3x3 Conv4	21.78	128 3x3 Conv4	18.84
[2 2] MaxPool2	0.02	[2 2] MaxPool2	0.036
512 3x3 Conv5	10.89	256 3x3 Conv5	9.42
512 3x3 Conv6	21.78	256 3x3 Conv6	18.84
[2 2] MaxPool3	0.01	[2 2] MaxPool3	0.01
8192x1024 FC1	10.89	4096x1024 FC1	18.84
1024x1024 FC2	1.36	1024x1024 FC2	4.71
1024x10 FC3	0.01	1024x10 FC3	0.04

map of size $E \times E \times N$. The kernels are flattened to single dimensional vector, and stored in the memory array into multiple rows, depending on the size of the flattened vector. For example, if the kernel size is, say, $3 \times 3 \times 64$, and the SRAM array has 64 columns, the kernel is stored in 9 rows. For Proposal-A, kernels can be mapped to different sections of the memory array, and can be computed in parallel, unlike in Proposal-B. Next, since these kernels stride over the input feature map E^2 times, the input feature map is split into these E^2 chunks of size $R \times R \times C$. Each of these chunks is flattened and stored in the array, similar to the flattening of the kernels. One of the chunks of the input feature map is shown in the figure as an example (in blue). Each Xcel-RAM array has some dedicated rows for storing these flattened activations (input feature map) and some for storing the flattened weights (kernels). We assume that this flattening and shifting is done offline, which we have not considered in our simulations. We assume that all data is stored in an off-chip DRAM, and we do consider the data movement required from the off-chip memory to Xcel-RAM banks, before the computations for a particular layer are performed.

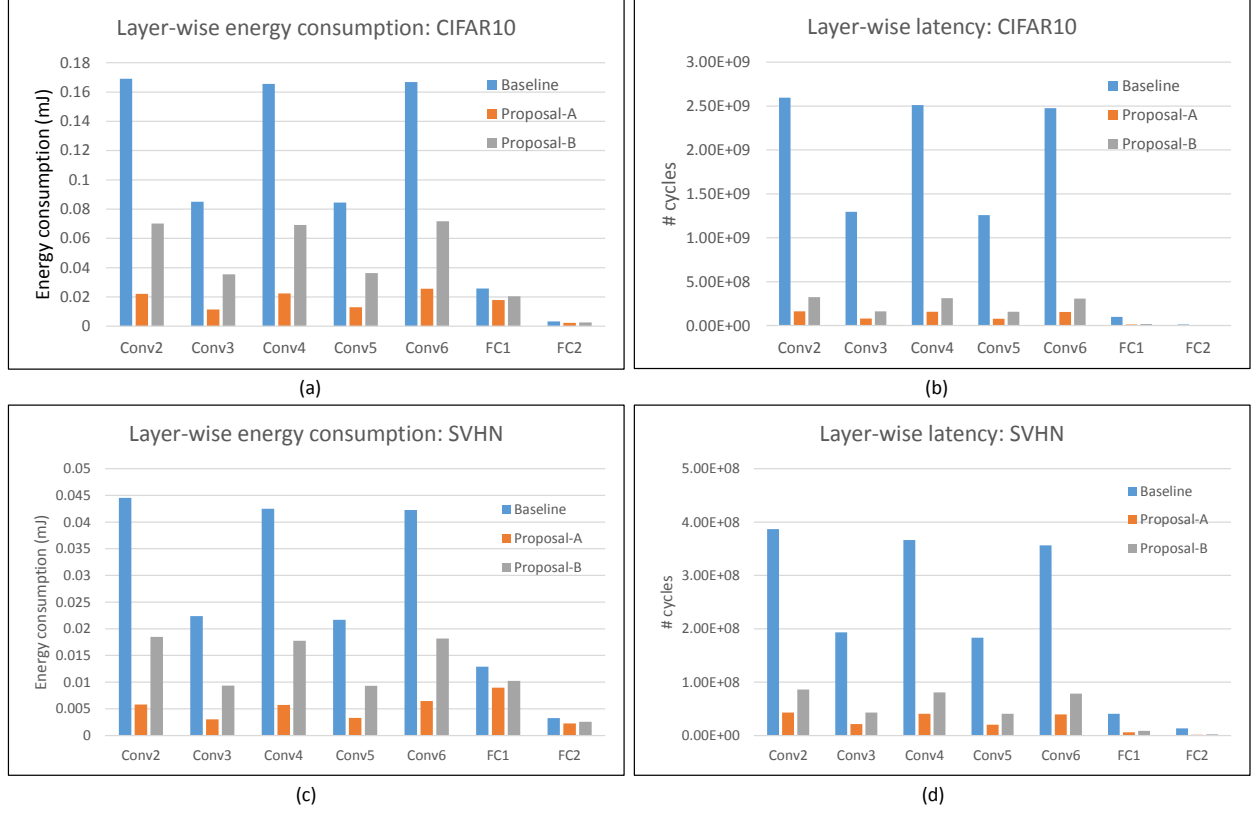


Figure 3.12. Layer-wise energy consumption and latency, for running the CIFAR-10 (a-b), and SVHN (c-d) image classification benchmarks on the proposed designs, and the baseline.

3.4.3 Results and Discussion

The accuracy of the binary neural network was observed to be 89.294% and 95.703% for CIFAR-10 and SVHN datasets, respectively. We then evaluate the impact of inaccuracies in the ADC for Proposal-A (due to process variations) on the classification accuracy using our simulation framework. At every binarized layer, each element of the output map is a sum of N binary XNORs, where $N = k^2 \times I$, k is the filter height, and I is the number of input channels. Our proposed methodology can perform 64 binary operations at once, in two steps of 32 bits each. Hence, the number of *popcounts* done per element of an output map is $M = \text{ceil}(N/32)$. We add the *popcount* error to the output during inference, obtained from circuit simulations, and obtained an accuracy of 88.710% and 94.129% for CIFAR-10 and SVHN, a decrease by 0.584% and 1.574% from the ideal BNN accuracy, respectively.

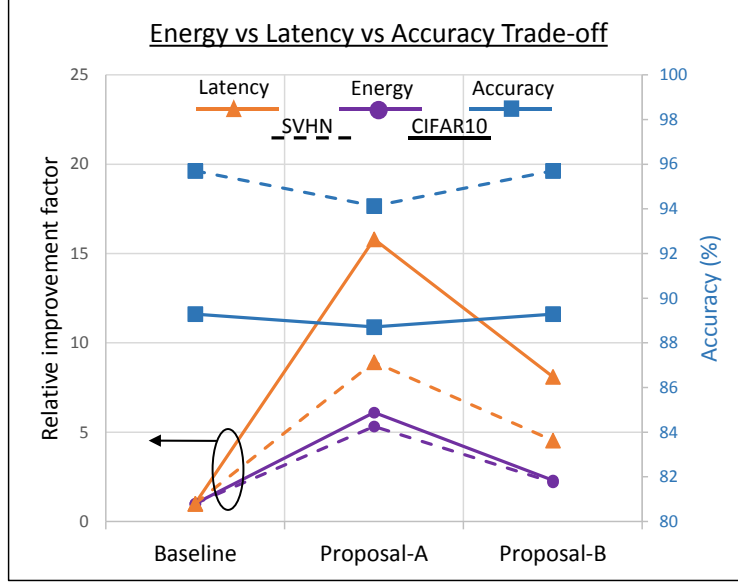


Figure 3.13. Energy, latency and accuracy tradeoff for classifying CIFAR-10 and SVHN dataset with BNN, using the proposed techniques.

On the other hand, Proposal-B obtains ideal BNN accuracy because the computations are done using a digital adder-tree.

Fig. 3.12 shows the layer-wise energy consumption and latency for Proposal-A, Proposal-B, and the baseline. Note that we focus only on layers Conv2-6 and FC1-2, as they constitute majority of the total computations. It can be observed that layers Conv2,4,6 are the most compute intensive layers, due to larger kernels. Overall, per-inference, $6.1\times$ and $2.3\times$ improvements were obtained in energy consumption, for Proposal-A and Proposal-B, respectively, compared to the baseline for CIFAR-10 dataset. In terms of latency, $15.8\times$ and $8.1\times$ improvements were obtained per-inference, for Proposal-A and Proposal-B, respectively. These improvements can be attributed to the fact that the most compute intensive operations involved in the BNN inference – bitwise-XNOR followed by *popcount*, are performed efficiently within the memory, thereby saving majority of unnecessary memory accesses and computations. Moreover, the energy and latency benefits of Proposal-A arise from the low-overhead ADC and the sectioned SRAM arrays, which enable multiple operations in a single memory access. In Proposal-B, although the sectioning is not applicable, the energy and latency benefits arise from the bit-wise XNOR computations on the bitline using asymmetric

SAs and the digital bit-tree adder to generate the result in the memory array itself. For the SVHN dataset, the energy improvements are $5.32\times$ and $2.20\times$, while the latency improvements are $8.92\times$ and $4.52\times$, for Proposal-A and Proposal-B, respectively, compared to the baseline. The improvements are lower compared to the CIFAR-10 implementation since the kernel sizes in the network used for SVHN are smaller compared to CIFAR-10. This shows that larger kernels translate to more effective utilization of the hardware primitives, leading to higher energy and latency improvements.

Figure 3.13 summarizes the tradeoff between energy, latency and classification accuracy for both the proposals. Proposal-A achieves large improvement factors for both energy and latency compared to the baseline. However, it has a reduced accuracy. On the other hand, Proposal-B has lower improvement factors, but achieves baseline accuracy.

3.5 Conclusion

Enhanced memory blocks having built-in compute functionality can operate as on-demand accelerators for machine learning computations, while simultaneously operating as usual memory read-write units for general-purpose workloads. We demonstrated two novel techniques to enable binary convolutions within a standard SRAM memory arrays. In the first proposal, we use charge-sharing on the inherent parasitic capacitances present in the 10T-SRAM structure to embed vector XNOR operations. Further, we use a dual-read wordline along with a dual-stage ADC, to handle the inaccuracies in the low precision, low-overhead ADC. A key highlight of this proposal is the *sectioned-SRAM*, which enables multi-row convolutions in parallel, thereby improving the overall system performance and energy-efficiency. The second proposal uses asymmetric SAs and a bit-tree adder in the memory peripherals to perform bit-wise XNOR computations and *popcount* in-memory. A complete framework was developed to evaluate benchmark applications (CIFAR-10 and SVHN) using our proposed memory arrays. For a system with our proposed Xcel-RAM banks, upto $6.1\times$ and $2.3\times$ improvements were obtained in energy consumption, and $15.8\times$ and $8.1\times$ improvements were obtained in the latency for the respective proposals, compared to conventional SRAM based system.

4. ENABLING DOT-PRODUCT COMPUTATIONS IN STANDARD 8T-SRAM ARRAYS USING CHARGE ACCUMULATION AND SHARING

4.1 Introduction

In the past decade, we have seen a tremendous growth in Machine Learning (ML) algorithms, especially Deep Neural Networks (DNNs). DNNs have been shown to be extremely effective for various cognitive applications, such as classification, recognition, detection and autonomous systems, which are being adopted into various disciplines [1], [2]. The primary reason for their exponential growth and widespread adoption in the past decade can be attributed to the advancements in computational power and resources [3]. Availability of powerful large-scale CPU and GPU servers and clusters enabled execution of computationally expensive DNN models, leading to superior performance [3]. Even today, the size of the state-of-the-art DNNs grows exponentially [4]. Although large-scale data-centers having multiple CPU clusters and GPUs, enable large parallelism and faster execution of DNNs, they are extremely power hungry. This is because DNN compute models are inherently different from general-purpose workloads and are immensely memory- and compute-intensive. Standard von-Neumann systems executing DNNs face the von-Neumann bottleneck [5], consuming more energy in frequent data movements than the computation itself [83]. This has largely restricted the execution of DNNs to large-scale data-centers, due to such high power demands.

Nowadays, most real-time data is generated at the edge-devices, such as sensor nodes, drones, and IoT devices. Most of these devices are battery-operated, and thus have limited battery life. Transferring large amounts of data from the edge devices to the cloud is not only energy expensive, but sometimes undesirable due to security reasons, such as in defense or automotive applications. Thus, there is a need for processing data at the edge, to enable energy-efficient DNN inference. There have been innovations in both algorithmic as well as hardware fronts, to mitigate the energy problem of exploding DNNs. Recently, there have been emergence of *memory-friendly* quantized networks, such as binary networks [6],

XNOR-Net [7], and ternary networks [8], [9]. The basic idea is to reduce the bit-precision of the network parameters (weights, or activations, or both), from full-precision (32-bit or 64-bit floating point) to low-precision fixed-point notation (1-bit, 2-bit, etc.). This drastically reduces the computational complexity of the network, while also reducing the amount of data movement, without significant loss in state-of-the-art accuracies owing to the error-resiliency of neural networks, and their ability to re-train. On the hardware side of things, there have been significant interest in beyond von-Neumann computing, especially the paradigm of in-memory computing [10]. This approach embeds some basic computations within the memory arrays, where the data is stored. By using such enhanced memory structures, frequent and unnecessary data-transfers between the memory and the compute units can be eliminated, without significantly changing the memory hierarchy and the conventional read/write functionality of memory arrays. Moreover, this opens up the internal bandwidth of memory arrays, which is much larger than the external input/output bandwidth, and can be exploited to enable parallelism.

Edge-devices have a very simplistic architecture with basic microcontroller units along with some on-chip caches. Standard on-chip caches use CMOS 6-Transistor Static Random Access Memories (6T-SRAMs) which are optimized for fast read and write operations. However, 6T-SRAMs have a shared read/write port, and are not suited for in-memory computing applications due to read-stability concerns. In 8T-SRAMs, there is an additional read port, thereby decoupling the read and write operations at the cost of two extra transistors. This provides additional SRAM stability, thereby enabling the possibility of in-memory computations [24]. Going to 9T- and 10T-SRAM cells improves the SRAM stability further, but at the cost of reduced storage density. Thus, we use standard 8T-SRAM cells as they provide a good balance between the in-memory computing functionality and storage density.

We propose an in-memory dot-product computation primitive using charge-sharing in 8T-SRAMs [84]. We show that inherent parasitic capacitances can be utilized for performing dot-product operations, where one operand is stored in the SRAM arrays while the other operand is applied as analog voltages on the BLs. The accumulated charge on the SL represents the desired dot-product output. However, the analog charge-domain nature of computations introduce circuit non-idealities that degrades the output accuracy. To that

effect, we further propose a self-compensation approach, wherein, the effects of these circuit non-idealities are reduced through a simple two-step procedure. In the first step, full V_{DD} is applied at the BLs and the SL voltage is sensed, while the dot-product operation is performed in the second step. The voltage sensed during the first step can be used to compensate the non-linearity in the second step. Using these compute primitives, we propose *CASH-RAM*, an in-memory computing primitive which can be integrated into on-chip caches for accelerating ternary weight neural networks. The key highlights are as follows:

1. We propose a compute primitive for performing dot-products based on charge-sharing using the inherent parasitic capacitances in standard 8T-SRAM arrays. We study the effects of non-idealities caused by analog computations, both analytically and through circuit simulations.
2. We propose an efficient two-step self-compensation approach to minimize the non-idealities while performing the dot-product computations. We demonstrate the effectiveness of the compensation technique, both, analytically and through circuit simulations.
3. We propose a system integration of the aforementioned in-memory computing primitive to enhance on-chip caches and augment them with compute instructions in addition to standard read/write instructions.
4. We develop a functional model of our proposed SRAM array and demonstrate an image classification application using the MNIST and CIFAR10 dataset to evaluate the accuracy degradation and to estimate the energy and throughput benefits of ternary weight neural networks.

The remainder of this chapter organized as follows. A summary of related works is provided in Section II. Next, the charge-sharing approach for in-memory dot product operation is described in Section III, followed by the self-compensation. In Section IV, we describe the system integration of the proposed computing primitive for accelerating ternary weight neural networks. Results on various networks for image classification tasks are reported in

Section V, including the functional accuracy, energy and throughput, before concluding in Section VI.

4.2 Related Works

While several digital ASICs have been explored for accelerating DNNs [21], [22], [85], they lack flexibility and cannot run general-purpose workloads. Thus, we will focus on in-cache computing based approaches, which achieve high performance for DNNs, while allowing flexibility of general purpose computing.

Digital In-memory Computing

In-cache computing has been explored in various types of SRAM bitcells, and for accelerating various operations. Bit-wise Boolean operations, such as NAND/NOR/XORs were initially demonstrated in standard 6T and 8T SRAM cells [24], [86], [87]. The basic idea is to enable multiple memory rows, and directly read out the resulting bitline (BL) voltage, which represents a Boolean logic operation of the data stored within the enabled rows. Adding further digital logic at the peripheral circuitry allows for more advanced arithmetic operations, such as for encryption [88] and DNN [68] processing.

Analog In-memory Computing

Analog-based computing provides possibilities for implementing more complex operations, with high degree of parallelism and energy-efficiency, compared to digital based approaches [10]. For example, applying a pulse-width modulated (PWM) signal at the wordlines (WLs) and sensing the analog voltage developed at the BLs, yields the dot-product operation between the input and the stored data, which has been explored in both 6T and 8T-SRAM cells [89]–[93]. However, PWM signal generation is very time sensitive, which can significantly be affected by the process variations. Some other works rely on compute primitives with efficient current-accumulation or charge-sharing approaches. In current-mode computations [62], [73], [94], the array is operated in a crossbar configuration, where the input voltages are applied at sourcelines (SLs) or WLs and the resulting current gets ac-

cumulated on the BLs proportional to the dot-product of the inputs and the data stored in the memory array. However, this requires expensive peripheral circuitry, that consumes large static power consumption [73]. Moreover, current-mode computations are sensitive to process variations, parasitic line resistances and electromigration.

Thus, a charge-domain compute seems to be a more viable option. In [72], [74], charge-sharing approach was used in 10T-SRAM cells to demonstrate binary convolution operation. In [72], the BLs were charged to the input voltages, and with a help of an additional switching circuitry, the charges were shared and accumulated. However, due to inaccuracies in interfacing conventional low precision DACs/ADCs, the work is limited to smaller networks. A more scalable approach was taken in [74], where the two-step WL activation yielded the convolution output inherently on the parasitic SL capacitance. However, it is difficult to generalize the computations to multi-bit dot-product operations. More recently, [95], [96] proposed a charge-domain compute in 8T-SRAMs, where a separate capacitor is attached to each bitcell to enable accurate charge-sharing. However, this requires fundamental modifications to the standard 8T-SRAM bitcell with an addition of an overlaying metal-oxide-metal (MOM) capacitor on each bitcell. On the other hand, [97] proposed to accumulate charge onto the computation and compensation capacitors that were attached to the peripheral circuitry, to perform the dot-product operation.

In contrast to previous works, we propose charge-sharing based dot-product acceleration in standard 8T-SRAM arrays, utilizing the inherent parasitic capacitances. Owing to the analog computations and the weak charge-sharing between parasitic capacitances, we analyze various errors that get introduced due to circuit non-idealities, both analytically and through simulations. Moreover, we propose at least two self-compensation schemes, which can be integrated seamlessly to improve the accuracy of the dot-product output.

4.3 Charge Sharing based In-Memory Dot-Product Operation

In this section, we describe the circuit details for the proposed charge-sharing based dot-product operation. As described in the introduction, we use the internal parasitic capacitance of the memory array to accumulate charge, which represents the desired output. We first

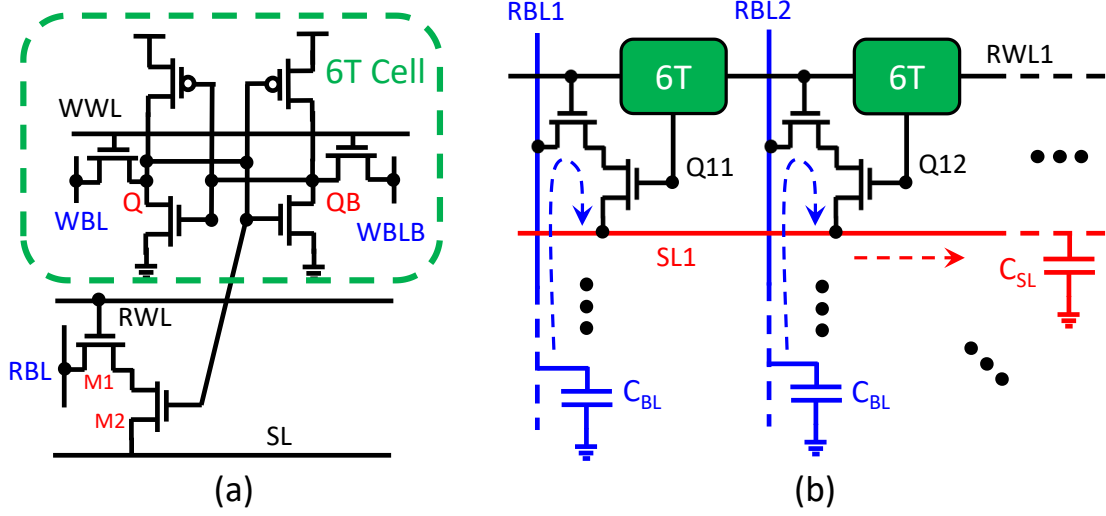


Figure 4.1. (a) Schematic of the standard 8T-SRAM bitcell (b) Parasitic capacitance C_{BL} and C_{SL} for an array of 8T-SRAM cells. The dotted arrows show the charge-sharing path used in our approach.

theoretically analyze the circuit operation through charge sharing equations, followed by SPICE simulations to characterize the circuit for dot-product output.

4.3.1 8T-SRAM: Structure and Operation

Fig. 4.1(a) illustrates the standard 8T-SRAM cell. In addition to the standard 6T cell, an 8T cell consists of an additional read port formed by transistors M1 and M2, connected to the read-wordline (RWL), read-bitline (RBL) and the source line (SL), as shown in the figure. Note that the SL is usually grounded for normal memory operations, however, with negligible penalty in the bitcell area, the SL can be routed horizontally in parallel to the RWL. The write operation of the 8T cell is exactly similar to the 6T cell. The write-wordline (WWL) is enabled for the row to be written, and the write-bitlines (WBL/WBLB) are given appropriate voltages ($V_{DD}/0$) depending on the data to be written into the internal storage nodes Q and QB . However, during the read operation, the additional read port is used, and not the write port. This makes the read operation much faster compared to 6T cells because the read-write ports are decoupled and the read port is optimized for the read operation. During the read operation, RBLs are pre-charged to V_{DD} , and the corresponding RWL is

enabled, while the SL is grounded. If the node Q of the cell stores a ‘1’, both transistor M1 and M2 are ON, thereby discharging the RBL through the SL. On the other hand, if the node Q stores a ‘0’, transistor M2 is OFF, thereby cutting off the discharge path such that RBL remains pre-charged. This dip in RBL voltage is sensed to read out the data from the cell.

Fig. 4.1(b) shows how the 8T cells are connected together to form a subarray. In a reasonable sized array, there is considerable parasitic capacitance developed on each of the BLs, SLs, and WLs running horizontally and vertically. These include the drain/gate capacitance of the transistors as well as the parasitic capacitance of the metal wire running along the array. For our purposes, RBL capacitance (C_{BL}) and SL capacitance (C_{SL}) are of particular importance, since they directly fall into the computation path, while other capacitances do not affect the dot-product functionality. For example, the write port capacitances (C_{WBL}/C_{WBLB} and C_{WWL}) are not utilized at all during the compute operation, while the RWL capacitance (C_{RWL}) is predominantly the gate capacitance of the access transistors, which only determines the access delay of the cell. C_{BL} and C_{SL} are highlighted in the figure in blue and red, respectively, and the dotted arrows show the charge-sharing path used in our approach. We propose to perform the dot-product operation by pre-charging the RBLs with analog voltage, and accumulating charge on the SL. This is described in the next subsection.

4.3.2 8T-SRAM: Charge sharing based Dot-Product Operation

A dot-product, also known as the inner dot product, is defined between two vectors say, \mathbf{X} and \mathbf{W} :

$$A = \sum_{i=1}^N x_i \cdot w_i \quad (4.1)$$

where, N is the length of the vectors \mathbf{X} and \mathbf{W} . This is a very frequent operation in ML workloads, especially neural networks. The vector \mathbf{X} represents the input activations while the vector \mathbf{W} represents the learned weights of the network. Note that the elements of vectors \mathbf{X} and \mathbf{W} can be full-precision (floating point), however, we will focus on hardware-friendly quantized networks, with ternary weights (+1/0/-1) and fixed-point input activations. This will be discussed in detail in Section IV. To perform this operation within the SRAM array,

we store \mathbf{W} in the SRAM cells, while \mathbf{X} is converted to analog voltages (\mathbf{V}) and applied to the RBLs. Please refer to Fig. 4.1(b). The first step is to pre-charge the RBLs with appropriate analog voltages, such that all C_{BL} are charged. The total charge in the system can be defined as:

$$Q_{ini} = C_{BL} \sum_{i=1}^N v_i \quad (4.2)$$

In the next step, the desired RWL is enabled, allowing the charge stored in C_{BL} to accumulate over C_{SL} . However, note that only those RBLs will interact which have the SRAM cell storing ‘1’, otherwise the discharge path is blocked by the read-port transistors. If we have say, K number of 1’s in the \mathbf{W} vector, then the initial charge (Q_{ini}) can be written as:

$$Q_{ini} = C_{BL} \sum_{i=1}^N v_i = \underbrace{C_{BL} \sum_{i=1}^N v_i \cdot w_i}_{K \text{ non-zero terms}} + \underbrace{C_{BL} \sum_{i=1}^N v_i \cdot \overline{w_i}}_{N-K \text{ non-zero terms}} \quad (4.3)$$

Thus, only the K non-zero terms are involved in charge sharing. The final charge of the system (Q_{final}) can be expressed as:

$$Q_{final} = V_{SL}(C_{SL} + KC_{BL}) + C_{BL} \sum_{i=1}^N v_i \cdot \overline{w_i} \quad (4.4)$$

where V_{SL} is the final voltage developed over the SL. The second term denotes the initial un-shared charge remaining on the RBLs. Through conservation of charge, $Q_{ini}=Q_{final}$, we can calculate the final SL voltage, V_{SL} as:

$$V_{SL} = \left(\frac{C_{BL}}{C_{SL} + KC_{BL}} \right) \sum_{i=1}^N v_i \cdot w_i \quad (4.5)$$

We can observe from Eq. 4.5, that V_{SL} is proportional to the dot-product of vector \mathbf{V} and \mathbf{W} , and thus, by sensing the analog voltage on SL we can estimate the dot-product.

4.3.3 SPICE Characterization

Next, we perform SPICE simulations to verify and characterize the analog voltage V_{SL} with respect to the inputs, v_i and w_i . We use a 65-nm PDK to simulate a 256×64 8T-SRAM

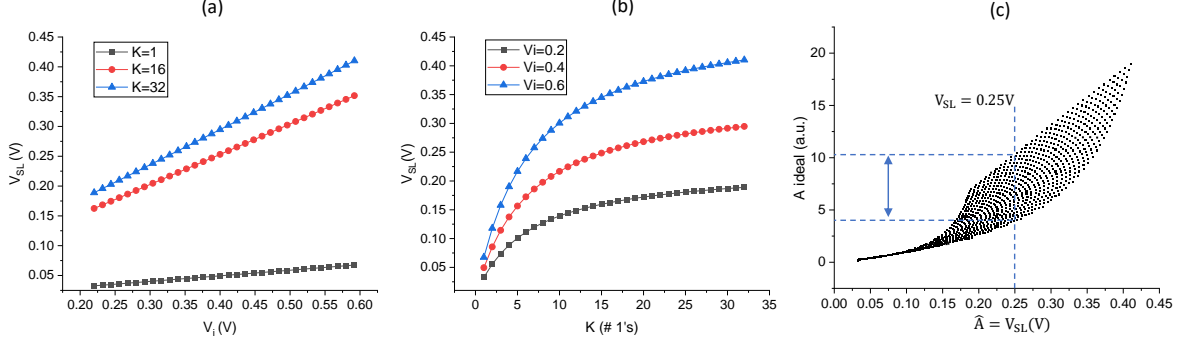


Figure 4.2. Data obtained from SPICE simulations. (a) Final SL voltage as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) Final SL voltage as a function of K . Three cases for $v_i=0.2,0.4,0.6$ V are shown. (c) Representation of V_{SL} as a function of ideal dot product A . The degree of spread represents the non-idealities, as illustrated in the figure taking an example of $V_{SL}=0.25$ V.

array which was illustrated in Fig. 4.1(b). The array was simulated for the aforementioned dot-product operation, with various combinations of v_i and w_i to see the effects on V_{SL} . There is a large number of possible combinations, thus, we perform simulations for some special cases, which are plotted in Fig. 4.2. We assume the same v_i value being applied to all RBLs. For example, the data point for $v_i=0.6$ V and $K=16$ implies a case where 16 cells store ‘1’ and participate in charge sharing, and all the RBLs are precharged to 0.6V. The plots in Fig. 4.2(a) show that V_{SL} linearly increases with the value of v_i . This follows from Eq. 4.5. However, in Fig. 4.2(b), we observe that V_{SL} increases linearly for small values of K , but starts to saturate as K increases. This also follows from Eq. 4.5, but is an undesirable artifact of the charge-based analog computing. This limits the maximum value of K . Our simulations showed that beyond $K=32$, the errors become too large. Thus, we process vectors of length 32 at a time for our dot-product micro-operation. More details regarding the architecture for our micro-operation will be discussed later in Section IV. In Fig. 4.2(c), we plot all of the data, with intermediate K and v_i values also, on a different axes, showing the estimated dot-product (\hat{A}) versus the ideal dot-product (A). This plot clearly shows how the error in the dot-product originates. Let’s say we obtain a $V_{SL}=0.25$ V after performing our dot-product operation. This voltage would be converted to \hat{A} . Now,

if we draw a vertical line through $V_{SL}=0.25V$, the scatter plot shows that this value can correspond to various A values. This is due to that fact that there are various possible combinations of inputs that yield the output as $0.25V$. Due to the non-linearity shown in the previous plots, each combination incurs different error characteristics, leading to the spread observed in the figure. Thus, there is a need to minimize this spread to reduce computation errors, which is explored next through a self-compensation approach.

4.3.4 Self-Compensation

To reduce the spread described above, we propose a two-step operation to perform the dot-product. The fundamental reason for this spread is the non-linearity introduced in Eq. 4.5 due to K , which is a data-dependent term and introduces data-dependent errors into the output. In our proposed self-compensation approach, we first try estimating the value of K in the first step, which is then used to compensate the dot-product output in the second step.

To estimate K , we perform the same computation as the dot-product above, but during the pre-charge phase, we first pre-charge all RBLs with V_{DD} , instead of the corresponding analog v_i values. Following from Eq. 4.5, the SL voltage in this step, V_{SL_1} will be:

$$V_{SL_1} = \left(\frac{C_{BL}V_{DD}}{C_{SL} + KC_{BL}} \right) \sum_{i=1}^N w_i = \frac{C_{BL}V_{DD}K}{C_{SL} + KC_{BL}} \quad (4.6)$$

Thus, K can be estimated by sensing the analog voltage V_{SL_1} :

$$\hat{K} = \frac{C_{SL}}{C_{BL} \left(\frac{V_{DD}}{V_{SL_1}} - 1 \right)} \quad (4.7)$$

In the second step, we perform the dot-product operation as usual, by applying analog v_i values at the RBLs, to obtain V_{SL_2} , similar to what was described in Eq. 4.5. However, substituting \hat{K} from Eq. 4.7 into Eq. 4.5, and solving for \hat{A} yields:

$$\hat{A} = \sum_{i=0}^N v_i \cdot w_i = \left(\frac{C_{SL}}{C_{BL}} \right) \frac{V_{DD}V_{SL_2}}{V_{DD} - V_{SL_1}} \quad (4.8)$$

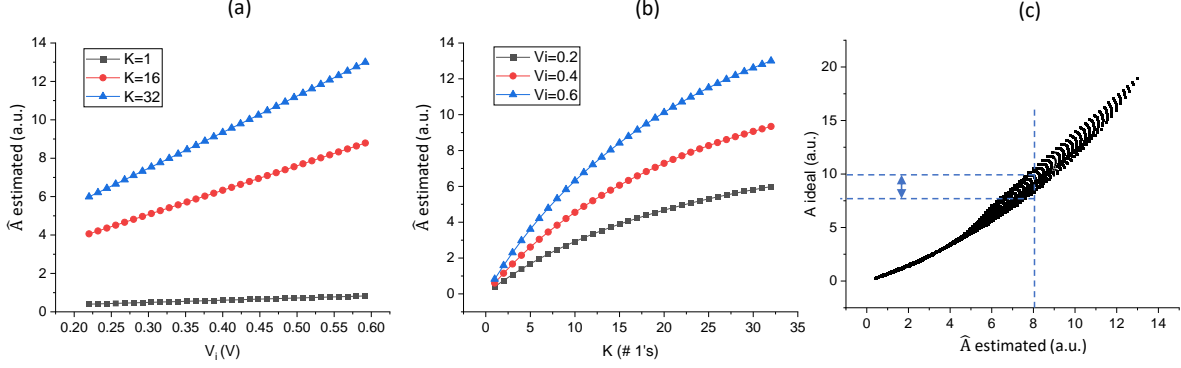


Figure 4.3. Data obtained from SPICE simulations and \hat{A} estimated from Eq. 4.8. (a) \hat{A} as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) \hat{A} as a function of K . Three cases for $v_i=0.2,0.4,0.6$ V are shown. (c) Representation of \hat{A} as a function of ideal dot product A . The degree of spread is significantly lower than in Fig. 4.2(c).

Eq. 4.8 summarizes our self-compensated charge-sharing approach to estimate the dot-product output based on sensing two analog voltages, V_{SL_1} and V_{SL_2} .

Next, we verify this approach through SPICE simulations. The two-step operation was simulated for the same 256×64 array, and analog voltages V_{SL_1} and V_{SL_2} were extracted. Moreover, other circuit constants such as C_{BL} and C_{SL} were also extracted from SPICE to estimate \hat{A} from Eq. 4.8. Fig. 4.3(a-b) plots \hat{A} as a function of v_i and K , respectively, similar to the plots shown in the previous section. We can observe a significant improvement in the linearity of Fig. 4.3(b), compared to Fig. 4.2(b), owing to the compensation mechanism in play in estimating \hat{A} . Similarly, in Fig. 4.3(c), we can observe a significant reduction in the spread in A versus \hat{A} , leading to reduced errors in estimating dot-product outputs in the self-compensated charge-sharing approach.

4.3.5 Compensating for Transistor Non-linearity

Until now, we have considered only ideal charge sharing among the parasitic capacitances C_{SL} and C_{BL} , and our hardware compensation approach also assumed ideal charge sharing equations. The source of non-idealities originates from transistors, which act as non-ideal

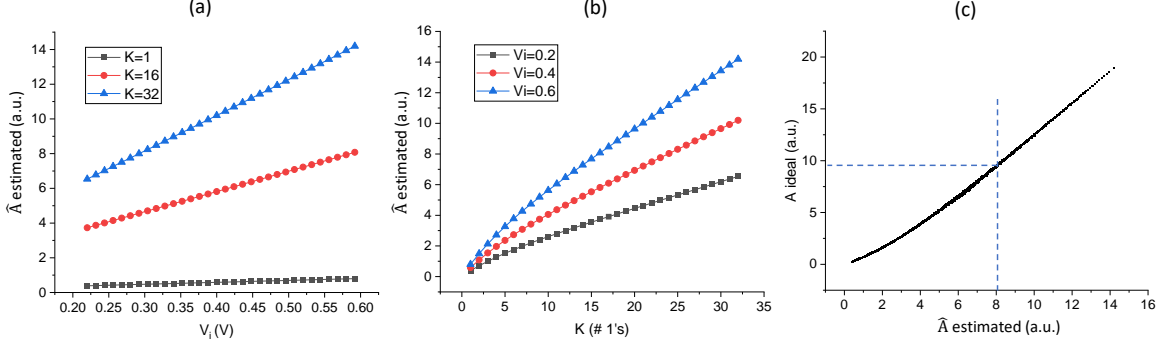


Figure 4.4. Data obtained from SPICE simulations and \hat{A} estimated from Eq. 4.9. (a) \hat{A} as a function of input voltage v_i . Three cases of $K=1,16,32$ are shown. (b) \hat{A} as a function of K . Three cases for $v_i=0.2,0.4,0.6$ V are shown. (c) Representation of \hat{A} as a function of ideal dot product A . The degree of spread has further reduced significantly than in Fig. 4.3(c).

switches. A few experiments with Eq. 4.8 revealed that a best fit was obtained if we combine V_{SL_1} and V_{SL_2} as follows:

$$\hat{A} = \sum_{i=0}^N v_i \cdot w_i = \left(\frac{C_{SL}}{C_{BL}} \right) \frac{(V_{DD} - V_T)V_{SL_2}}{\sqrt{V_{DD} - V_T - V_{SL_1}}} \quad (4.9)$$

where V_T is the threshold voltage of the transistors. The two main differences between Eq. 4.8 and Eq. 4.9 is the V_T term and the square root in the denominator. From transistor physics, we understand that an NMOS connected to V_{DD} at its drain and gate terminal can only charge up its source node to $V_{DD} - V_T$, because it enters cut-off region. Thus, the maximum charge that can be shared is $V_{DD} - V_T$. On the other hand, the square root term acts as an additional non-linear compensation to improve the linearity of the circuit. This can be seen from Fig. 4.4(a-c), which illustrates the improvement in the linearity of the estimated dot product, and the reduction in the spread using this compensation technique. Note that these plots were obtained from the same data collected for V_{SL_1} and V_{SL_2} as before, however, they were combined using Eq. 4.9 instead. Hence, we classify this approach as an additional compensation to the previous approach. However, we would also like to mention here that there might be more complex equations to combine V_{SL_1} and V_{SL_2} that yield even better results, however, as the equation gets more complex, it would require more circuitry

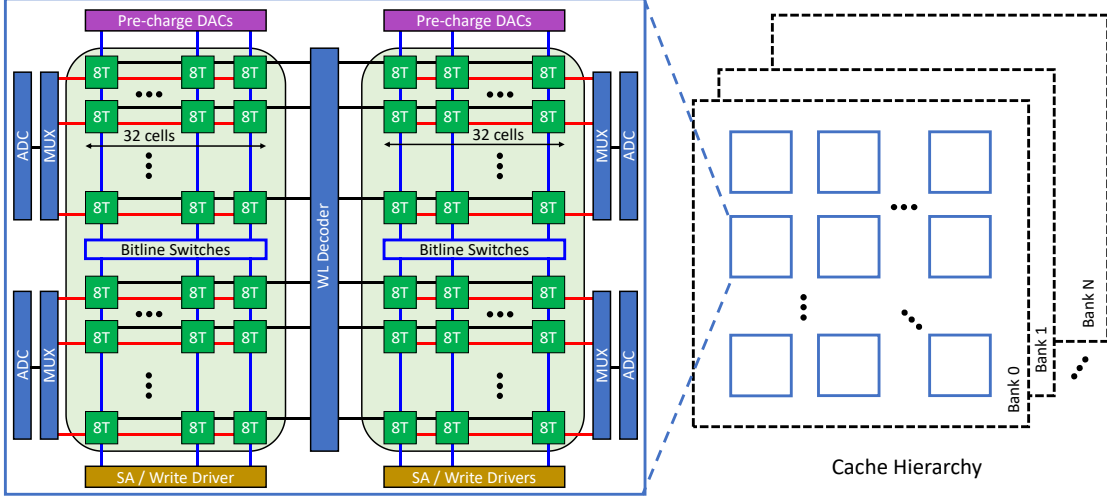


Figure 4.5. CASH-RAM system integration for accelerating TWNNs. A typical multi-bank cache hierarchy is shown on the right, where each bank consists of multiple sub-arrays. The subarray is shown on the left, with additional peripheral circuitry to augment dot-product computations within the cache.

or look-up tables to compute the result, eliminating the benefits of accelerating dot-product operations in the first place. Thus, it is evident, that there is a trade-off between complexity and accuracy of obtaining the dot-product through this approach, which is beyond the scope of this work.

4.4 System Integration of CASH-RAM for Accelerating Ternary Weight Neural Networks

In the previous section, we described how standard 8T-SRAM cells can be used for computing approximate dot product operations. Dot product operations constitute majority of the computations in DNNs. Moreover, among quantized DNNs, ternary weight neural networks (TWNNs) represent a promising tradeoff between benefits due to model compression and accuracy [98], [99]. Thus, in this section, we take our compute primitive further, and propose a cache integration for accelerating TWNNs.

4.4.1 Cache Integration

The caches in a modern processor are organized in a hierarchical structure with multiple banks, as shown in Fig. 4.5. Each bank consists of multiple sub-arrays. To perform in-situ dot-product operations, the cache can be repurposed as an on-demand accelerator, where each subarray can perform parallel computations [68]. Using the standard cache hierarchy, we modify the sub-array with some peripheral circuitry to augment the dot-product operations, in addition to the standard memory read/write instructions. Before the cache can begin the neural network processing, the weights for different layers are loaded in from off-chip memory. However, the positions of the weights do not change and remain stationary in the cache. Thus, the weights are loaded only once per layer, thereby amortizing the overhead over the duration of the neural network run. In this re-purposed cache, each subarray performs the exact same operation in parallel and is capable of generating the outputs independently.

4.4.2 Subarray Details

An expanded version of the subarray is shown on the left in Fig. 4.5. It consists of standard 8T-SRAM cells arranged in two halves with 32 columns each. As discussed in Section 4.3, sharing charge among more than 32 cells resulted in unacceptable errors, thus, we restricted the number of cells in a row to 32. This was done by splitting the SLs of the 256×64 array in two, such that the SLs are only shared among 32 cells in each half (shown in red horizontal lines), while the WLs connect all 64 cells in a row (shown in black horizontal lines). The sub-array consists of the necessary peripherals, including the pre-charge circuits, WL decoder, write drivers, and sense amplifiers to support traditional memory read/write operations. Additionally, pre-charge circuits are enhanced with digital-to-analog converters (DACs) to charge up the RBLs, and analog-to-digital converters (ADCs) to sense the SL voltages, during the dot-product operation. The write ports of the 8T SRAM have been omitted in the figure for clarity. Furthermore, to increase parallelism of the dot-product operations, we introduce bitline switches, thereby dividing the sub-array into sections. For example, with 256 rows in the sub-array, we may have bitline switches (shown in blue vertical lines) every 32 rows, thereby having 8 sections. This was also earlier proposed in [72], [74],

[100]. During the pre-charge phase of the dot-product operations, the switches are closed, and hence the entire RBL gets pre-charged. Once the RBLs are pre-charged, the switches are opened, separating the pre-charged RBL sections each of which can independently perform the dot product operation by sharing the charge with its local SLs. Each section has its own ADC, thus, with a single pre-charge, multiple operations can be performed simultaneously.

4.4.3 Data Mapping

The weights of a pre-trained TWNN can be mapped to the proposed cache arrays. The network weights or kernels in TWNNs can be 0 or ± 1 . However, since SRAM bit stores a 0/1, we use a differential architecture proposed in many previous works [101] to map both positive and negative weights to SRAM cells. In a differential form, a ternary weight w can be represented as $w = w^+ - w^-$. Thus, w^+ and w^- are either 0/1, and can be easily represented in the SRAM bitcells. The input activations of the TWNN are applied as voltages to the RBLs to compute the dot-products. We assume fixed point representation for the input activations, which can be routed through the data lines into the sub-array. The pre-charge DACs convert the fixed-point digital values to an analog voltage on the RBLs. In this case, the dot-product computations become:

$$A = \sum_{i=1}^N v_i \cdot w_i = \sum_{i=1}^N v_i \cdot w_i^+ - \sum_{i=1}^N v_i \cdot w_i^- \quad (4.10)$$

Since w^+ and w^- require the same set of activations to perform the dot-product, alternate sections separated by the bitline switches (refer Fig. 4.5) can store w^+ and w^- , respectively. The resulting digital output from the two sections can be subtracted to obtain the dot-product output.

4.5 Results

In this section, we analyze the benefits of the proposed macro for image classification workloads. Firstly, we analyze the effects of errors induced in performing the dot-product operations due to analog computing, on the classification accuracy for various benchmarks.

Secondly, we estimate the energy and throughput benefits of running such workloads on the proposed macro, compared to the von-Neumann baseline and other prior works.

4.5.1 Experimental methodology

All circuit simulations, which were presented in Section 4.3 were performed using the 65-nm PDK in H-SPICE. As described earlier, various combinations for v_i and K were simulated to analyze the error profile due to non-idealities. A Monte-Carlo analysis was done with 1000 runs for each of the combination of (v_i, K) . The analog values for V_{SL_1} and V_{SL_2} were recorded in each case. We observed that the 3σ deviation in V_{SL} values due to variations was well within 1-LSB error of the ADC, leading to negligible affect on the actual output. The operating range of v_i was chosen to lie in $v_i \in [0.2V, 0.6V]$, while $K \in [0, 32]$ to minimize the error range.

The collected simulation data was characterized and plotted on an error surface spanning the v_i and K ranges. The error in the ideal output (A) and the estimated output obtained from the simulations (\hat{A}), for each of the approaches described in Section 4.3, was mathematically formulated using a best curve-fit.

To estimate the impact of these errors on the overall system accuracy, the obtained error functions were transferred to a software implementation in Pytorch. The errors were added to each layers of the ternary networks during the forward pass, by breaking up the computation into smaller dot-products of length 32, and introducing the error functions into each of these smaller operations. Thus, for a deep network, these errors accumulate over multiple operations, and we expect an accuracy degradation, which is explored next.

4.5.2 Impact of Non-idealities on Classification Accuracy

In order to comprehensively analyze the impact of the previously discussed error compensation approaches, we perform experiments with networks and datasets of varying complexity. Image classification datasets are most generic due to their widespread usage and easily available benchmarks. Thus, we employ the MNIST handwritten digit dataset [102], and the CIFAR-10 image dataset [103]. The MNIST dataset having images of handwritten digits (0-

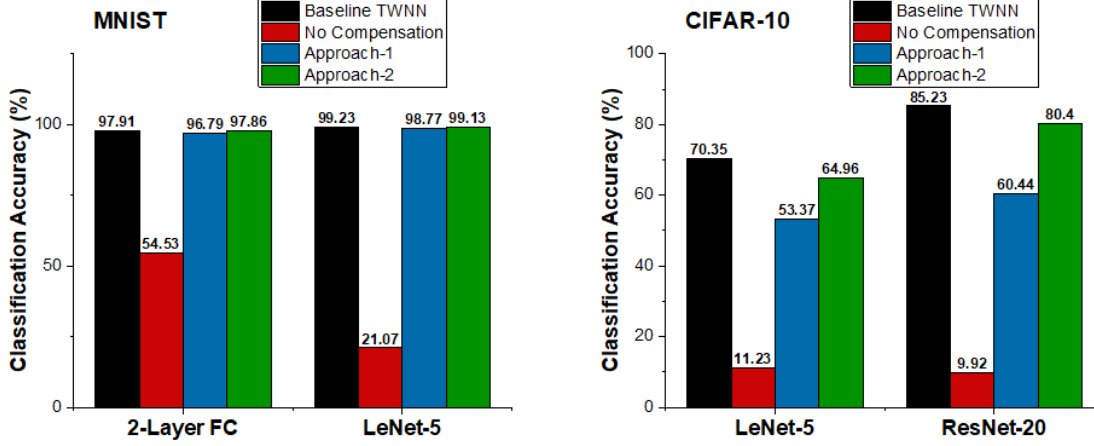


Figure 4.6. Classification accuracy obtained on MNIST and CIFAR-10 datasets for different benchmarks, for with and without self-compensation. Approach-1 corresponds to self-compensation while Approach-2 corresponds to compensating for transistor non-linearity.

9) is comparatively easy to train even with a network having only linear fully-connected(FC) layers. This helps in verifying that the effects of error addition are very minimal when the task at hand is relatively simple. On the other hand, CIFAR-10 is a very popular real image dataset having 10 different classes and is complex enough to require usage of convolutional layers to obtain satisfactory level of accuracy. In terms of neural networks, we chose to have a mix of different depths to observe the impact of errors as we go from shallow to deep networks. Keeping these points in mind, we considered the following image classification benchmarks for our analysis:

1. MNIST: 2-layer FC Network, LeNet5 [104]
2. CIFAR-10: LeNet5 [104], ResNet-20 [105]

The networks were trained without bias and with batch-norm at full-precision initially. This was followed by training with ternary weights while still maintaining full-precision gradients. This ternarized model served as the baseline for our analysis. The accuracy values for each of the compensation approaches as well as the baseline were compared and are highlighted in Fig. 4.6.

We can observe that performing the dot-product naively without compensation degrades the classification accuracy to unacceptably low values in all cases. This was expected, considering the large errors in the dot-product output as shown in Section 4.3. For the simpler MNIST dataset, we observed that self-compensation worked quite well, and the accuracy drop is within 1% of the baseline. However, for a more complex dataset, CIFAR-10, the accuracy drop from the baseline was higher ($\sim 17\%$ for LeNet5 and $\sim 24\%$ for ResNet20). With the additional transistor non-linearity compensation, the accuracy drop was significantly improved to within $\sim 5\%$ in both cases. This can also be directly attributed to the more accurate dot-product estimation using the compensation approaches, described in Section 4.3.

Note that this analysis only involves error addition and inference on the test-set without any further training of the error added models. There is further room for improvement with re-training approaches [27], [29], [106]–[108], where the neural network training includes the error added models in the forward pass. However, our approach is complementary to the re-training approach and can be used in tandem with such approaches to improve the overall system performance further. An additional analysis revealed that re-training the error added model of LeNet-5 network for CIFAR-10 dataset for just 1 epoch, reduced the accuracy drop further to within $\sim 3\%$.

4.5.3 Energy, Delay and Area Estimates

We analyze the energy and performance benefits of the proposed primitive based on the approach mentioned in [92], [93], followed by estimates on the area overhead. In order to evenhandedly compare our approach with these works, we study the energy and delay of our implementation on LeNet5 and ResNet20, when inferred with MNIST and CIFAR10 image, respectively, with memory and architectural specification similar to [92]. We have N_{bank} number of banks, each having N_{arr} arrays of size $N_{row} \times N_{col}$; B_{IO} represents the number of bits fetched by CPU in single cycle; number of bits of weights is given by B_w ; and R represents the number of row-wise parallel operation enabled by the BL switches. The values chosen for these parameters are summarized in Table 4.1. Further, we adopt the von-

Neumann computing framework as the baseline. The energy and delay for the von-Neumann system represented by E_{VN} and T_{VN} , respectively, are given by Eq. 4.11 and Eq. 4.12 [93]:

$$E_{VN} \approx MNK^2 E_{read} + MNK^2 N_{mov}^2 E_{mult} + MNN_{mov}^2 E_{reg} + P_{leak} T_{VN} \quad (4.11)$$

where E_{VN} is the energy of von-Neumann architecture, E_{read} is the read energy of a single weight from SRAM array, E_{mult} is the energy for multiplication energy in the processor, E_{reg} is the energy required to store/load the weights in the CPU registers, P_{leak} is the average leakage power of the SRAM and T_{VN} is the time required for von-Neumann compute given by equation 4.12. The software parameters M , N , K , L and N_{mov} represent the number of input channels, number of output channels, kernel size, size of input feature map and size of output feature map, respectively, for the layer under consideration.

$$T_{VN} \approx \left[\frac{MNK^2}{(B_{IO}/B_W)N_{bank}} \right] T_{read} + \left[\frac{MNK^2}{N_{mult}} \right] N_{mov}^2 T_{mult} \quad (4.12)$$

here, T_{read} refers to the time required to read weight from the SRAM array and T_{mult} is the delay for multiplication operation in the processor.

The energy and delay estimates for our primitive, represented by E_{CS} and T_{CS} , are given by Eq. 4.13 and Eq. 4.14 respectively:

$$E_{CS} \approx MNK^2 N_{mov}^2 B_W \left[\frac{E_{comp}}{32} + \frac{2E_{adc}}{32} \right] + MNN_{mov}^2 E_{reg} + P_{leak} T_{CS} \quad (4.13)$$

where E_{CS} is the energy of our in-memory compute scheme, E_{comp} is energy required to compute 1 multiply and accumulate, E_{adc} is the energy required by ADC and T_{CS} is the time required for von-Neumann compute given by Eq. 4.14.

$$T_{CS} \approx \left[\frac{MNK^2}{(N_{col}/B_W)N_{arr}N_{bank}R} \right] N_{mov}^2 [\max\{T_{comp}, 2T_{adc}\}] \quad (4.14)$$

Table 4.1. Hardware Parameters Description

Parameter	Description	Value
Architectural Parameters		
B_{IO}	Bits fetched from SRAM per bank	16
B_W	Bit-width of the weight stored in SRAM	2
N_{col}/N_{rows}	Number of columns/row in SRAM array	64/256
N_{arr}	Number of arrays in SRAM bank	8
N_{bank}	Number of SRAM banks	4
N_{mult}	Number of multipliers in processor	175
R	Number of row-wise parallel operations	8
Delay Parameters		
T_{read}	Time to fetch data from SRAM	4.0 ns
T_{mult}	Time to perform 1 MAC in processor	1.0 ns
T_{comp}	Time for 1 MAC in our proposed array	3.0 ns
T_{adc}	Time for 1 ADC operation	4 ns
Energy Parameters		
E_{read}	Energy to fetch weight from SRAM	1.3 pJ
E_{mult}	Energy to perform 1 MAC in processor	225 fJ
E_{comp}	Energy for 1 MAC in our proposed array	360 fJ
E_{adc}	Energy for 1 ADC operation	231.1 fJ
P_{leak}	Standby power consumption of SRAM	2.4 nW

here, E_{comp} is time required to compute 1 multiply and accumulate and E_{adc} represents the time required by ADC for conversion.

We follow a pessimistic approach and compare our results for the input conditions leading to maximum energy E_{comp} . Moreover, we know E_{comp} should directly depend on the charge drawn from supply Q_{VDD} , hence maximizing Q_{VDD} should maximize the energy E_{comp} . For the case of self-compensation, Q_{VDD} should be dependent on K and v_i as per Eq. 4.15. This equation is maximized for max K and min v_i , hence we find the E_{comp} and T_{comp} for these inputs using H-SPICE simulation. The architecture parameters were kept same as [93], the

Table 4.2. Network Parameters Description

Parameter	Description
M	Number of input feature maps
N	Number of output feature maps
K	Kernel size
L	Size of input feature map
N_{mov}	Size of output feature map ($N_{mov}=L-K+1$)

Table 4.3. Energy, Delay, and Area Comparison

Property	Baseline	[93]	[92]	This work
SRAM cell	8T	6T	6T	8T
Input/Weight Precision	5b/2b	5b/5b	6b/8b	5b/2b
Area overhead	0%	56%	25.31%	68.25%
MNIST on LeNet5				
Energy (nJ)	466.52	451.10	330.38	302.28
Delay (μ s)	10.06	1.53	13.94	0.4
EDP (fJ-s)	4695.03	688.08	4606.75	122.58
CIFAR10 on ResNet20				
Energy (μ J)	28.87	34.87	22.97	20.20
Delay (μ s)	268.52	150.38	1374.73	40.12
EDP (pJ-s)	7751.43	5244.37	31579.04	810.09

energy and delay numbers for ADC are obtained from [93]. All the parameter values are summarised in Table 4.1 and Table 4.2.

$$\frac{Q_{VDD}}{C_{BL}} = K \left[V_{DD} - \left(\frac{KC_{BL}}{KC_{BL} + C_{SL}} V_i \right) \right] + (N - K)(V_{DD} - V_i) + \max \left\{ K \left[V_i - \left(\frac{KC_{BL}}{KC_{BL} + C_{SL}} V_i \right) \right], 0 \right\} \quad (4.15)$$

The energy, delay, and the Energy Delay Product (EDP) for inference of one MNIST image is presented in Table 4.3, along with a comparison with the baseline and the prior works. We find that *CASH-RAM* is 38 \times better in EDP compared to the von-Neumann baseline with similar architectural parameters. If we scale EDPs by the ratio of bit precision

Table 4.4. Area breakdown

Supporting Circuit	Area per 256×64 Array (μm^2)
Array	13845
ADC	12800
DAC	50
MUX	250
Bitline switches	525
Decoder	600
Precharge, SA, and Driver	5500

of weight and activation linearly, we find that *CASH-RAM* is $\sim 7.8\times$ and $\sim 2.25\times$ better than [92] and [93] respectively.

The results for CIFAR-10 on ResNet-20 are also shown in Table 4.3. We can observe that the energy benefits for ResNet-20 on CIFAR-10 is $\sim 9.5\times$ which is a significant reduction as compared to $\sim 38\times$ on MNIST on LeNet-5. ResNet-20 has 19 convolution layers and 1 fully connected layer, while LeNet-5 has 2 convolution and 3 fully connected layers. The reduction in the improvements is observed due to the weight sharing in convolution layers. As a result, in a von-Neumann architecture the processor can reuse the weights for several computations which reduces the number of read operations dramatically, assuming a low-level cache or register files. Such reuse of weights is not possible in in-memory functional read as the read output is a function of both weights(W) and inputs(V_{in}). These results concur with the results shown in [92] and [93].

Let us now discuss the area overhead to implement our proposed *CASH-RAM* in caches. The area breakdown for a subarray of 256×64 is presented in Table 4.4. The additional peripheral circuitry required for our proposal is the ADC, DAC, MUX, and bitline switches, which constitute about $\sim 40\%$ of the total area. We can observe that after the array, the most dominant component is the ADC. Thus, to minimize the area overhead of ADCs, we adopted the ADC from [93], and scaled it down to 4-bit precision. To justify the 4-b precision ADC for our 32-vector operation, we tested the impact of this quantization loss on the classification accuracy, and made sure it was within $\sim 1\%$ of the full-precision accuracy.

4.6 Conclusion

Implementation of DNN workloads have been restricted to large-scale data centers, due to extremely high compute and energy requirements. To bring cognitive computing to the edge, quantized DNNs, such as TWNNs have been proposed which drastically simplify computations and reduce the model sizes. Moreover, domain-specific hardware primitives such as in-memory computing have been shown to realize energy-efficient implementations for DNNs. We proposed an in-memory computing primitive in standard 8T-SRAM arrays for computing dot-product operations, which are extensively used in DNNs, using inherent charge sharing. The estimated dot-product is approximate, owing to the analog nature of computing. Thus, we study and analyze the sources of these errors and propose a self-compensation to mitigate the effects of circuit non-idealities and non-linearities. Using this computing primitive, we proposed an 8T-SRAM macro for accelerating TWNNs. We develop a functional simulation framework to characterize the induced errors and estimate the degradation in classification accuracy of TWNNs, and to estimate the energy and latency of inference tasks. We demonstrate that using the proposed compensation approaches, the classification accuracy degradation is within 1% and 5% of the baseline accuracy, for the MNIST and CIFAR-10 dataset, respectively, with an EDP improvement of $38\times$ over the von-Neumann baseline. The proposed techniques are complementary to existing mitigation techniques, such as re-training, and can be used in conjunction with such approaches to further improve the overall system performance.

5. LOOKUP TABLE BASED COMPUTING USING ROM-EMBEDDED SRAM

5.1 Introduction

Rapid advancements in computing technology has enabled widespread development of increasingly large and complex models throughout various fields of science and technology. Such models heavily use large math tables, high-order polynomials, transcendental function evaluations *etc.* One of the fast ways to efficiently compute such large function evaluations is through the use of on-chip read-only memories (ROMs). However, large dedicated on-chip ROMs incur large area overhead. This impairs the chip-floorplan, leading to increased delays. Moreover, while evaluating workloads which do not require on-chip ROM accesses, the part of the chip that is dedicated for ROMs lies waste. Thus, large ROM data such as math tables, truth tables, transcendental functions, high-order polynomials *etc.* are stored off-chip on cheap, but slow, memories, thereby leading to large access delays.

Another way to incorporate ROMs on-chip is by embedding them within on-chip memories, or caches, so that the chip-area is not wasted. There are a few earlier attempts found in the literature which combine on-chip RAMs and ROMs. In [109], multiple bit-lines are routed to the SRAM cells, and the ROM data is read by accessing each bit-line. However, the read latency of SRAM doubles and write power increases by $\sim 30\%$ due to increased bit-line parasitic capacitance. In [110], multiple power-supply rails are selectively connected to SRAM bit cells to embed ROM data. In [111], [112], additional transistors are added to the bit cells for the same purpose. Thus, all the above works incur larger area and/or latency. Recently, [113] demonstrated embedding ROMs within conventional SRAM caches on-chip, without degrading the area/performance penalty on RAM accesses. The idea was to use two WLs that run in parallel for each row in the memory array. Each bit-cell stores the RAM data as usual, but depending on which WL it is connected to, the ROM data is embedded. However, the ROM retrieval process destroys the RAM data stored in the bit-cell. Thus, for every ROM access, a temporary copy of the RAM data is made and then stored back after the ROM data is retrieved. This incurs high energy and performance overheads for reading ROMs. Moreover, during the ROM retrieval mode, there is a 5T-write into the SRAM cell,

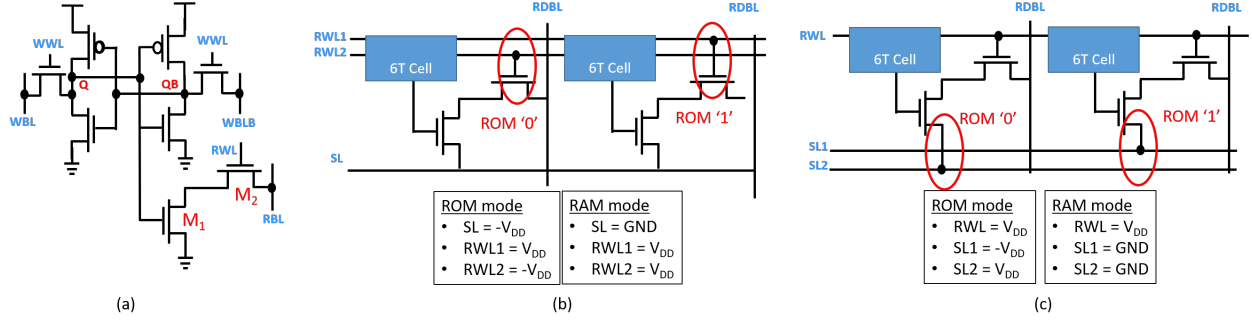


Figure 5.1. (a) Schematic of a standard 8T-SRAM bitcell. Transistors M1 and M2 form the decoupled read port of the SRAM cell. (b) Proposal-A for RECache. The cell has an extra RWL, and the ROM data stored is ‘1’ if the access transistor is connected to RWL1, and ‘0’ if it is connected to RWL2. The node voltages for RAM and ROM modes of operation are listed. (c) Proposal-B for RECache. This configuration has two SLs, instead of two RWLs. ROM data stored is ‘1’ if the access transistor is connected to SL1, and ‘0’ if it is connected to SL2.

thereby increasing the probability of write-errors. Some other works also propose embedding ROMs in non-volatile memories, such as Magnetoresistive RAMs (MRAMs) [114], [115], by adding an additional bit-line (BL) and sensing circuitry.

We propose ROM-embedded cache (RECache) [116], which embed ROMs within standard 8T-SRAM cells. Our approach addresses both the concerns raised above. First, 8T-SRAM cells have a decoupled read/write path, which enables a read-disturb free operation. Thus, we use the read-port of the 8T cell to embed ROMs, without affecting the stability of the RAM data stored in the cell. This enables a non-destructive readout of the ROM data, while preserving the RAM data in the bit-cell, unlike previous works. Moreover, the ROM retrieval process is very similar to the conventional RAM read access and does not require writing into the bit-cell, thereby improving the ROM access latency and energy consumption. We also show how to embed ROMs in a differential configuration of the 8T-cell given by [45], which further improves the storage density of the RECache. We also evaluate RECache on realistic workloads, like neural networks, cryptography etc. and understand why RECache performs extremely well for ‘ROM-intensive’ workloads.

The 8T-SRAM structures have recently been shown successful in performing Boolean [61] as well as non-Boolean [44] computations. Thus, in order to expand the scope of RE-

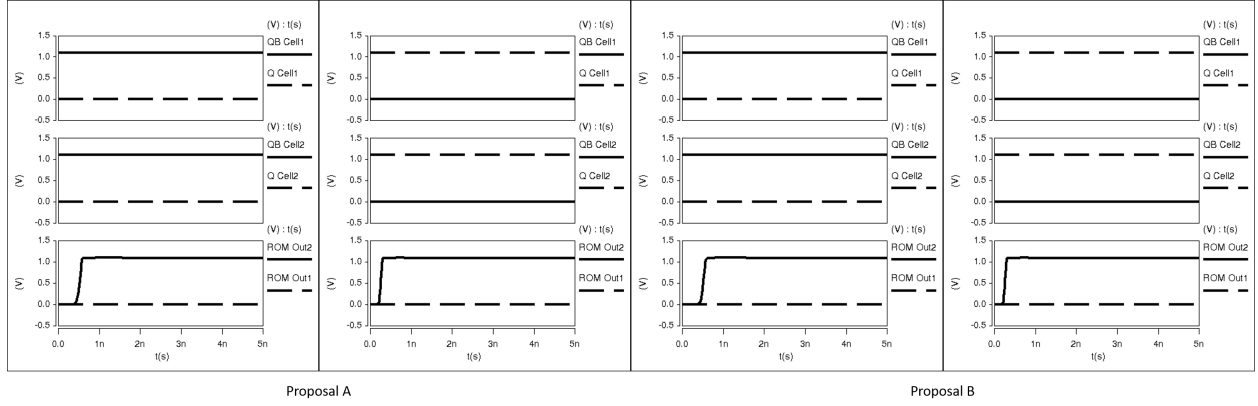


Figure 5.2. Timing diagrams generated from HSPICE for Proposal-A,B. The correct ROM Output data is generated, without disturbing the RAM data stored in the cell. The circuit shown in Fig. 5.1(b-c) are simulated.

Cache, we demonstrate its applicability for in-memory computations. Using the framework for spiking neural network (SNN) acceleration with ROM-embedded RAMs, developed in [117], we demonstrate the energy and performance benefits of RECache over conventional SRAM caches. The rest of the chapter is organized as follows. Section II describes the design and operation of RECache. In Section III, we evaluate RECache performance on realistic workloads, emphasizing its benefits for ROM-intensive applications. In Section IV, we describe the implementation of an SNN accelerator using RECache, before concluding in Section V.

5.2 RECache: Design and Operation

5.2.1 8T-SRAM

Fig. 5.1(a) shows the circuit schematic for a standard 8T-SRAM bit-cell. It consists of two transistors M1 and M2 in addition to the 6T storage cell. The two extra transistors form a decoupled read-port of the bit-cell. The write operation is similar to the 6T-cell, through the write-port (WWL, WBL, WBLB). However, the read-port is utilized (RWL, RBL, SL) for performing read operations. RWL and SL are connected to V_{DD} and GND, respectively, while RBL is initially pre-charged to V_{DD} . If the bit-cell stores ‘1’ (*i.e.*, $Q = '1'$), the charge on RBL discharges. If the bit-cell stores a ‘0’ (*i.e.*, $Q = '0'$), RBL holds its pre-charge

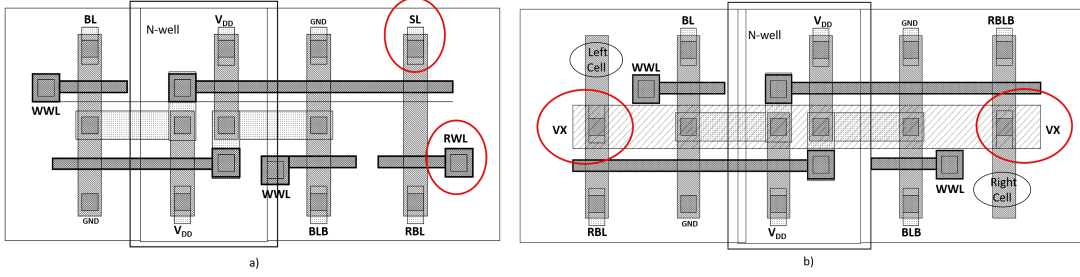


Figure 5.3. (a) Thin cell layout of a standard 8T-SRAM cell. The circled contacts, RWL and SL, are common to adjacent bitcells in the horizontal and vertical direction, respectively. (b) Thin cell layout of the 8⁺T-SRAM bit-cell. The circled contacts (VX) on either side of the bitcell are shared by adjacent bitcells.

voltage. The voltage at RBL is sensed using a sensing circuit. The decoupled read-write port for the 8T cell allows a read-disturb free operation. Thus, large voltage swing (almost rail-to-rail) on the RBL can be maintained for easier sensing.

In Fig. 5.1(b) and Fig. 5.1(c), we show two proposals (Proposal-A and Proposal-B) to embed ROMs in 8T-cells, using an extra RWL and an extra SL, respectively. In Proposal-A, the transistor M2 of the bitcell is connected to either RWL1 or RWL2, depending on whether the ROM data to be embedded is ‘1’ or ‘0’, respectively. The write-port remains the same as in the standard 8T cells. Note that if RWL1 and RWL2 are connected, the circuit is exactly the same as the 8T-SRAM cell. Thus, for reading the RAM data, we follow the standard read procedure, with RWL1 and RWL2 both turned ON. To access the ROM data, only RWL1 is turned ON, while RWL2 and SL are pulled down to $-V_{DD}$. The transistor M2 of the bit-cells will be enabled only if it is connected to RWL1. Moreover, transistor M1 of all the bit-cells in the row will be enabled, irrespective of whether $Q = ‘1’$ or ‘0’ since the SL is at $-V_{DD}$. As a result, the RBL starts to discharge towards $-V_{DD}$ if the ROM stores ‘1’, and remains charged at V_{DD} , if the ROM stores ‘0’. This ROM data can be read by sensing the voltage on RBL using the same sensing circuitry. Note that although the internal RBL swings from V_{DD} to $-V_{DD}$, the sensing amplifiers are still driven by V_{DD} , thus the output swing is limited from 0 to V_{DD} . Also note that retrieving the ROM data does not destroy the RAM data stored in the bit-cells.

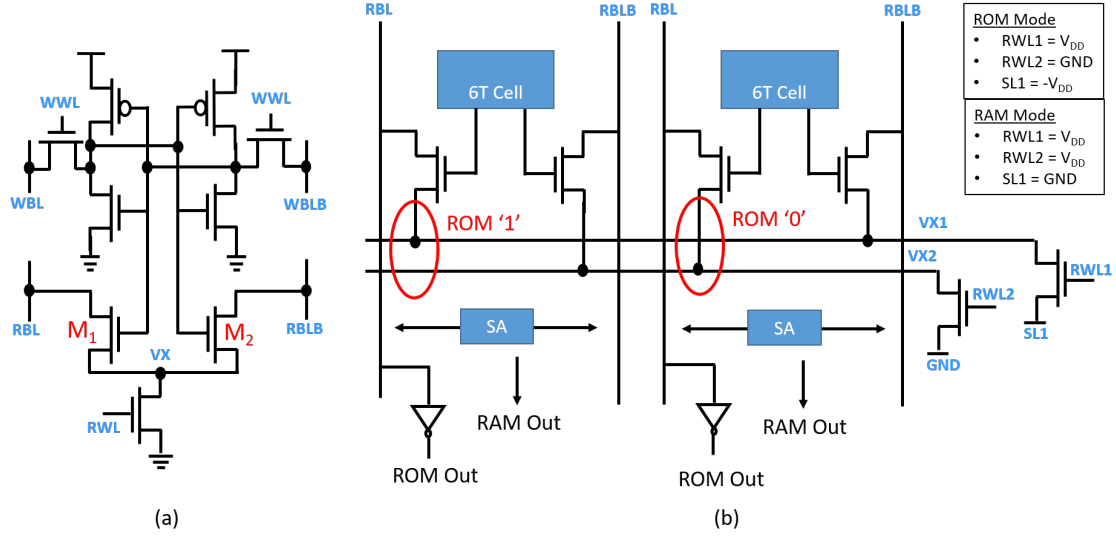


Figure 5.4. (a) Schematic of the differential 8⁺T-SRAM bitcell. Transistors M1 and M2 form a differential read port decoupled from the 6T write port. The ninth transistor connected to RWL is common for the entire row. (b) RECache using the differential 8⁺T-SRAM. The connection of M1 to either VX1 or VX2 determines the ROM bit stored in the cell. The ROM retrieval process is exactly same as the 8T-RECache. The node voltages for ROM and RAM mode of operation are listed.

In Proposal-B, instead of having two WLs, we have two SLs – SL1 and SL2, as shown in Fig 5.1(c). During the RAM mode of operation, both SLs are connected to the ground, thereby following the usual read operation of the 8T-SRAM cell. Now, the bit-cell connected to SL1 stores ROM data ‘1’, while the bit-cell connected to SL2 stores ROM data ‘0’. In the ROM mode of operation, SL1 is pulled down to $-V_{DD}$, while SL2 and RWL are pulled up to V_{DD} . Thus, RBL discharges only if the bit-cell is connected to SL1. Sensing the RBL voltage through the same sensing circuitry as before gives out the ROM data. Note that this proposal is also non-destructive since the RAM data stored by the bit-cells remains undisturbed during the ROM mode of operation, similar to Proposal-A.

Fig. 5.2 shows the transient simulations for Proposal-A and Proposal-B, obtained from SPICE, verifying the RAM and ROM mode of operation. The circuit shown in Fig. 5.1(b-c) was simulated in two scenarios – RAM data stored in the bit-cells is ‘1’ and ‘0’. The timing diagrams verify that ROM Output is correctly generated, irrespective of the RAM data

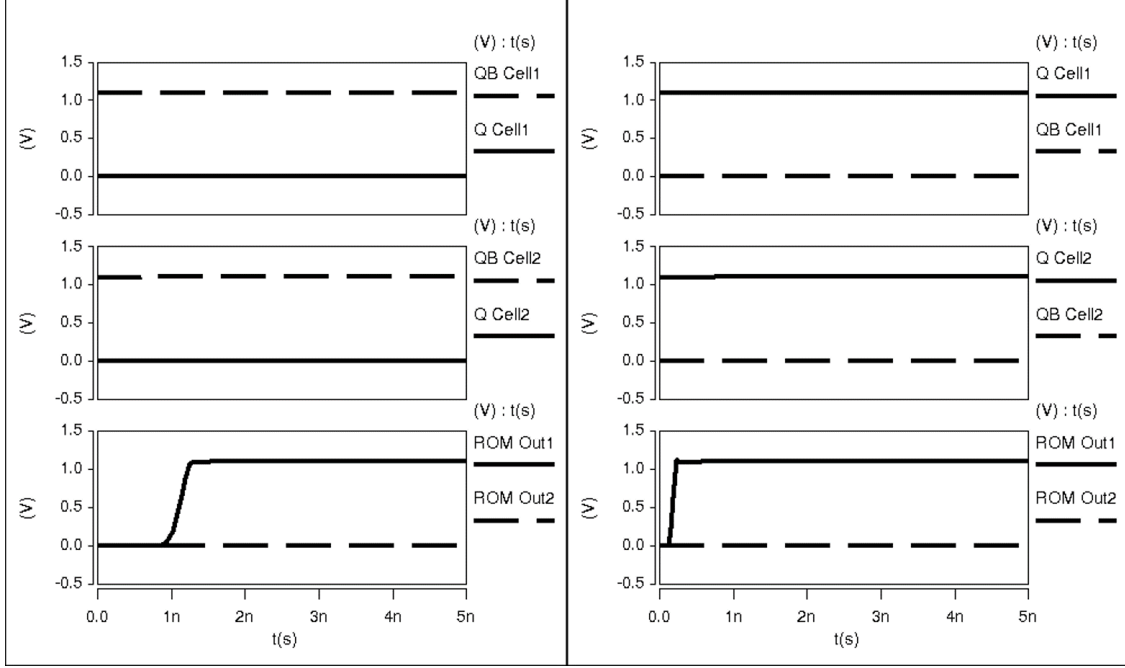


Figure 5.5. Timing diagrams obtained from HSPICE. The correct ROM Output data is generated, without disturbing the RAM data stored in the cell. The circuit shown in Fig. 5.4(b) was simulated.

stored in the bit-cells. Moreover, the RAM data is undisturbed during the ROM retrieval process.

5.2.2 8^+T Differential Read SRAM

In the previous section, we described two proposals to embed ROMs within 8T-SRAM arrays, thereby enhancing the memory density. However, there is one disadvantage of the proposal. The ROM storage is only half of the total RAM storage of the array. This is due to the layout restrictions of the 8T-SRAM bit-cell. For Proposal-A, there are two RWLs that run parallel, and the bit-cell is connected to either based on the ROM data to be embedded. However, two consecutive bit-cells share a common contact to the RWL, as shown in Fig. 5.3(a), and hence cannot store separate ROM data. Assuming an 8T-SRAM array of size 64×64 , we only have 64×32 bits of ROM. Similarly, for Proposal-B, we will have 32×64 ROM bits, since two consecutive bit-cells in the vertical direction share a common contact for the SLs (encircled in Fig. 5.3(a)).

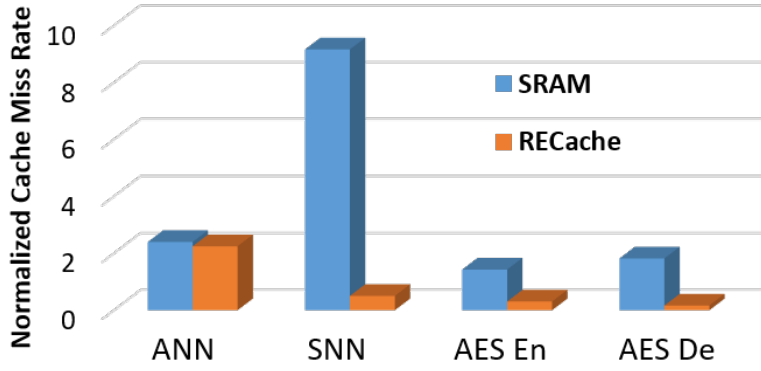


Figure 5.6. Normalized cache miss-rate for various benchmarks – artificial neural network (ANN), spiking neural network (SNN) and advanced encryption standard (AES), with RECache and standard SRAM caches.

Recently, an 8^+ T Differential SRAM design was proposed in [45] to overcome the single ended sensing of the conventional 8T-SRAM cell. 8^+ T Differential SRAM has decoupled read-write paths with an added advantage of a differential read mechanism through the read bit-lines RBL/RBLB (see Fig. 5.4(a)), as opposed to the single-ended read mechanism of 8T-SRAM. The ninth transistor, whose gate is connected to RWL is shared by all the bit cells in the same row. The differential read operation is very similar to the read operation of a standard 6T-SRAM. The usual memory read operation is performed by pre-charging the bit-lines (RBL and RBLB) to VDD, and subsequently enabling the word-line corresponding to the row to be read out. Depending on whether the bit-cell stores ‘1’ or ‘0’, RBL or RBLB discharges. The difference in voltages on RBL and RBLB is sensed using a differential sense amplifier.

We show how ROMs can be embedded within differential 8^+ T cells while maintaining 1:1 ratio between RAM and ROM density. The bit-cells are connected to either of the two nodes VX1 and VX2 which are connected to the ninth common transistor gated by RWL1 and RWL2, respectively. Again, two consecutive bit-cells share a common contact for the VX node (see Fig. 5.3(b)), however, due to a differential read port, the consecutive bit-cells can store different ROM bits. Consider an example shown in Fig. 5.4(b). The ROM data stored is ‘1,0’, which is decided by whether transistor M1 is connected to VX1 (ROM = ‘1’)

Table 5.1. Benchmarks used to evaluate RECache [49], [118]

Benchmark	Details	Peak memory required
ANN	Artificial Neural Network (784 x 1200 x 1200 x 10) for MNIST dataset	2.41MB
SNN	Spiking Neural Network (784 x 1200 x 1200 x 10) for MNIST dataset	2.54MB
AES	128b/256b-Key CBC/CTR/ECB mode encrypt/decrypt	64KB

or VX2 (ROM = ‘0’). During the ROM mode of operation, node VX1 is pulled down to $-V_{DD}$, while VX2 is kept floating. Thus, all RBLs corresponding to the bit-cells storing a ROM bit ‘1’ discharge. We use the single-ended sensing similar to the one used in 8T-cells, to sense the RBL voltage. The RBLB voltage is not considered for ROM operation. During the RAM mode of operation, both VX1 and VX2 are pulled down to 0V, and the differential SA is enabled. Thus, the differential SAs are used in the RAM mode to generate RAM Output, while the single-ended sensing of RBL is used in the ROM mode, to generate ROM Output. Fig. 5.5 shows the transient simulations obtained from SPICE, verifying the RAM and ROM mode of operation.

5.3 Evaluating RECache on realistic workloads

To understand the scope and the benefits of using RECaches as a substitute to the usual SRAM caches, we evaluate the cache performance for various workloads. The details of various benchmarks used are detailed in Table 5.1. Since our aim is only to show the benefits of extra ROM storage provided by RECache, we use a simplistic cache configuration: 32KB direct-mapped cache, with a block size of 64B. For each of the standard benchmarks in Table 5.1, an address trace was generated, and fed to a simple in-house cache simulator, to estimate the number of cache miss/hits. Since RECache has extra ROM storage, all ROM accesses are modeled as hits. Note that the cache configurations were chosen such that all the ROM data required by the program fits in the cache. Fig. 5.6 plots the cache miss rates of various

Table 5.2. ROM and RAM energy per-access for various array sizes obtained from CACTI.

Energy (nJ)	16KB	32KB	64KB
ROM Access	0.0224	0.0245	0.028
Read Access	0.0094	0.0115	0.015
Write Access	0.0109	0.0134	0.0175

8T RECache

Energy (nJ)	16KB	32KB	64KB
ROM Access	0.0224	0.0245	0.028
Read Access	0.0064	0.0085	0.012
Write Access	0.0098	0.0131	0.0185

8+T RECache

workloads for RECache as well as standard SRAM cache. The plots can be explained as follows. We notice that for an ANN, the RECache hardly makes a difference to the miss rate ($1.06\times$ improvement). This is because in each ANN layer, computations follow a sequential RAM-ROM address trace, ie, all ROM accesses (neuron activation) occur after all RAM accesses (synapse computations). Moreover, in all typical ANN applications, number of synapses are orders of magnitude more than the number of neurons. Thus, there is minimal contention between ROM and RAM data in the cache, thereby nullifying the benefits of RECache. However, RECache performs much better with ‘ROM-intensive’ workloads such as SNNs ($19.4\times$ improvement) and AES encrypt and decrypt ($4.6\times$ and $11\times$ improvement, respectively). This is because 1) these workloads use more frequent ROM accesses and 2) the address traces for ROM and RAM accesses are interleaved, thereby causing contention between RAM and ROM data in cache. SNNs involve multiple transcendental function evaluations and bio-realistic differential equations with high order polynomials, heavily relying on ROMs for fast computations. Similarly, in AES encryption/decryption, there are numerous substitution tables and multiplication math tables which are stored in ROMs and frequently accessed. Thus, RECache improves cache performance for workloads with high ROM vs RAM utilization and high ROM/RAM access interleaving. Note that for all other applications, RECache performs at least as well as standard SRAM caches.

5.4 Conclusions

With an ever-increasing demand for ROM-based computing, there is an imminent need for area-efficient ROMs on-chip. To that effect, we proposed ‘RECache’ – ROM-Embedded 8T-SRAM cache. RECache maintains the area and performance of standard SRAMs. Moreover, ROM retrieval process is non-destructive, ie., RAM data remains undisturbed during ROM accesses. We presented at least three different variants of RECache, giving designers options to choose from, as the application demands. Moreover, we demonstrated that RECache improves the cache miss rate by up to $19.4\times$, for ‘ROM-intensive’ workloads, which have frequent ROM accesses interleaved with RAM accesses.

6. SPIKING NEURAL NETWORK ACCELERATION USING LOOKUP TABLE BASED IN-MEMORY-COMPUTING

6.1 Introduction

Deep Neural Networks (DNNs) are inspired from the hierarchical learning behavior in the human brain and have tremendously enhanced the learning capabilities in machines [1], [2]. They have been credited to achieve high performance across a variety of recognition applications, even surpassing human abilities in certain tasks [51]. In doing so, however, DNNs tend to consume orders of magnitude higher energy than the human brain. To bridge this energy gap, there have been proposals from the algorithm as well as hardware perspectives. Spiking neural networks (SNNs), or third generation neural networks, have evolved and have been shown to achieve comparable classification accuracies with respect to the non-spiking counterparts [118]. SNNs rely on transfer of neuron spikes from one layer to the next, resembling the information transfer in the human brain. These spikes are encoded as binary data, thereby drastically simplifying the computations, and thus reducing the energy consumption.

On the other hand, hardware systems running DNN algorithms are inefficient, since DNN executions are memory- as well as compute-intensive. For instance, AlexNet which won the ImageNet 2011 challenge consists of 61 million parameters and involves 2-4 GOPS per classification [55], [119]. Consequently, their execution on von-Neumann machines consumes more energy for data movement than computation [119]. This can be attributed to the fact that DNN computation is inherently different from the conventional von-Neumann based computing model. Frequent data movement between a physically separate memory storage unit and a compute core forms the well known von-Neumann bottleneck. To overcome this bottleneck, there has been intense research for reducing data movements [64]. Moreover, there have been proposals for *in-memory computing* [13], [15], where the underlying principle is to perform the computations as close to the memory as possible, or better, within the memory array itself [69], [120], [121].

Typical DNNs (with artificial neurons) involve multiple transcendental function evaluations (for instance, sigmoid, tanh, logarithms etc.). In addition, SNNs involve several

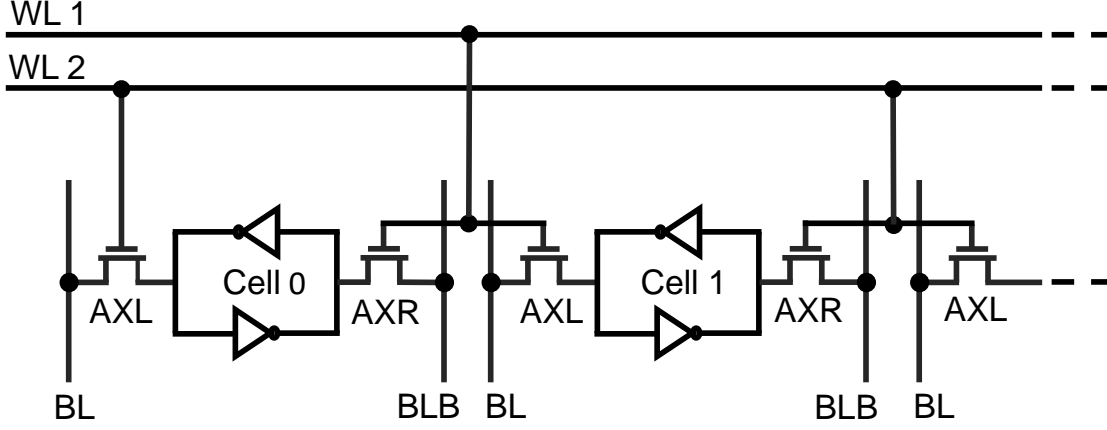


Figure 6.1. R-SRAM Schematic: Standard 6T-SRAM embedded with ROM. The only difference is the addition of extra word-line (WL1 and WL2) to embed ROM functionality.

bio-realistic neuron and synaptic differential equations, each having multiple transcendental function and high order polynomial evaluations. The most efficient way to implement such functions is by storing look-up tables (LUTs) and math tables in read-only memories (ROMs). However, large dedicated ROMs incur significant area and power overheads. To that effect, [113] proposed embedding ROMs in standard CMOS SRAM caches (R-SRAM). R-SRAM allows placing a ROM within the conventional SRAM array (with corresponding architectural modifications), without degrading the area and performance benefits of the SRAM [113]. Such compute primitives provide significantly higher storage densities (bits/area) which can be leveraged for DNN and SNN computations in storing useful data (LUTs) without affecting the RAM storage, thereby avoiding longer latencies and higher access energy associated with larger (or external) memory structures.

We take R-SRAMs and R-MRAMs a step further and propose “SPARE” [117], a generalized architecture for SNN acceleration using ROM-embedded RAMs as in-memory-compute primitives. SPARE consists of a 2-D array of Processing Elements (PEs) that spatially map a deep SNN, where each PE performs part of the SNN computations. Each PE contains its own R-SRAM/R-MRAM which locally stores only the relevant synaptic data and the LUTs required for solving the neuron and synaptic differential equations. This localized processing leads to energy benefits, since only the neuron data (spikes) need to be transferred

between PEs. Furthermore, since the PE operates only on an occurrence of an input spiking event, unnecessary computations and memory accesses are avoided. It is also worth noting that R-SRAM/R-MRAM primitive can store several different neuron and synapse models, thereby providing necessary flexibility. A PE thus, synergistically combines the hardware benefits from R-SRAMs/R-MRAMs and algorithmic benefits from SNNs. In summary, we make three key contributions.

1. **We design an energy-efficient PE** that leverages the “in-memory processing” abilities of ROM-Embedded RAM structures and “event-driven computing” in SNNs. We evaluate the pros and cons of using both, CMOS based R-SRAMs and STT-MRAM based R-MRAMs, as memory units in the PE.
2. **We design an efficient architecture (SPARE)** using a 2-D mesh of PEs, to provide a platform for cognitive application deployment. We show the implementation of spiking neural networks (fully connected and convolutional) on SPARE.
3. **We investigate the energy, performance and area benefits** for typical image classification benchmarks to underscore the system scalability and utility, both for training and inference phases.

6.2 Background

6.2.1 ROM-Embedded RAMs

Previous studies on ROM-embedded RAMs were limited to logic testing and fast mathematical function evaluations [113], [115]. We explore the utility of R-SRAM and R-MRAM based memory structures towards designing efficient compute primitives for neuromorphic computing (SNN acceleration). Further, as discussed before, such memory units enable in-memory data processing that can be of immense utility in DNN execution, which are typically limited by the cost of data movements.

R-SRAM

R-SRAM is a memory structure that consists of a ROM in hardware embedded into a conventional CMOS SRAM array, with corresponding modifications at the architectural

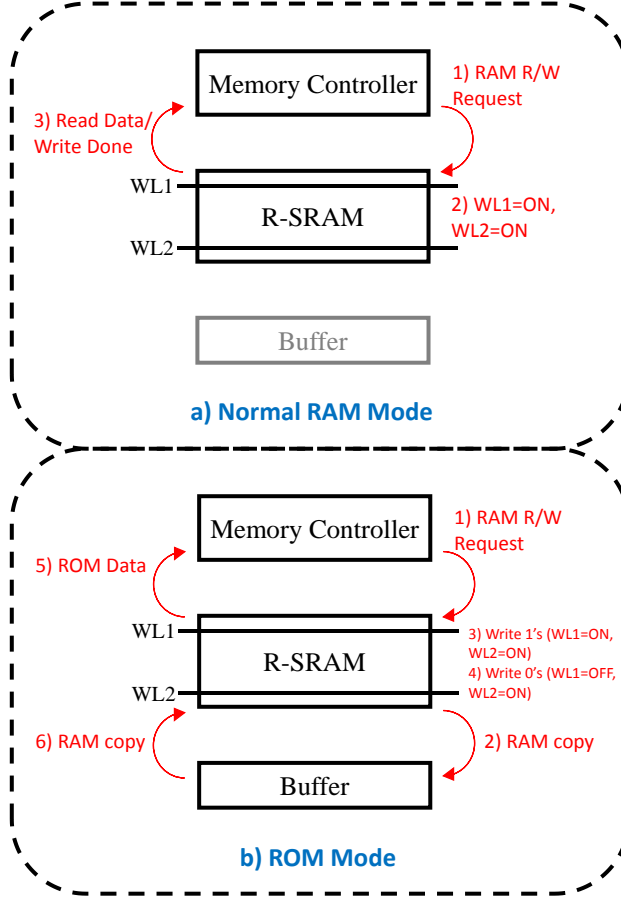


Figure 6.2. Operation of R-SRAM in a) Normal RAM Mode and b) ROM Mode.

level to support ROM accesses [113]. Fig. 6.1 shows the structure of R-SRAM cell array [113]. Unlike conventional 6T-SRAMs, R-SRAMs bit cells have an extra word-line (WL). The gate of the access transistors connect to WL1 or WL2, depending on the data to be embedded as ROM. Thus, if the bit-cell stores ‘0’ (‘1’) as ROM data, the left access transistor (AXL) is connected to WL2 (WL1). The right access transistor (AXR) of the bit-cell follows the connectivity of the AXL of the neighboring bit-cell to the right. For completeness, we describe the R-SRAM operation, both for the RAM mode and the ROM mode of operation.

1. *RAM mode:* During the normal RAM mode, both word-lines, WL1 and WL2, are connected together. They are turned ON/OFF at the same time, so as to operate as conventional 6T-SRAM for memory read/write. Note that there is no performance penalty on RAM operations compared to the standard 6T-SRAM bit-cells.

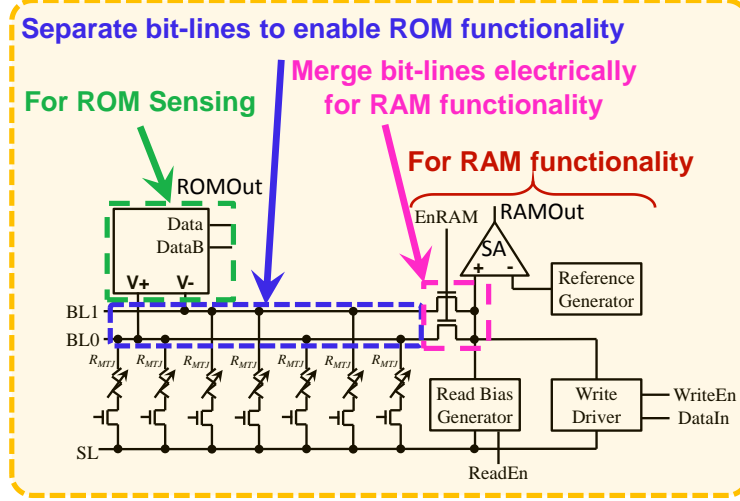


Figure 6.3. R-MRAM Schematic: Standard STT-MRAM array with two bit-lines (BL1 and BL0) to embed ROM functionality. The peripheral circuitry for RAM and ROM mode of operation is highlighted.

2. *ROM mode:* To retrieve the ROM data in the ROM mode of operation, a sequence of steps are performed, summarized in Fig. 6.2. First, ‘1’ is written to all bit-cells by turning both WL1 and WL2 ON. Thus, the whole row stores “1111...”. Next, WL1 is turned OFF and ‘0’ is written to all the cells, while WL2 remains ON. Now only the bit-cells connected to WL2 store ‘0’, others store ‘1’. However, if two consecutive bit-cells have different ROM data, this step performs a 5T write operation on the SRAM cell, since only one access transistor is ON. This may lead to a “write stability” problem in the bit-cells, which can be resolved using write-boost techniques [113]. The ROM data can now be read using conventional RAM read operation. Note that the ROM data retrieval process destroys the initial RAM content. Hence, before ROM data retrieval, RAM data of the corresponding block is written into a buffer, as shown in Fig. 6.2. After the ROM data has been retrieved, the RAM data of the block is restored.

It has been shown that R-SRAM incurs insignificant area ($\sim 2\%$) and power ($\sim 1\%$) overheads [113] to incorporate an additional word-line requirement. Moreover, we will show later in our simulations that despite the penalty of buffering RAM data for each ROM access,

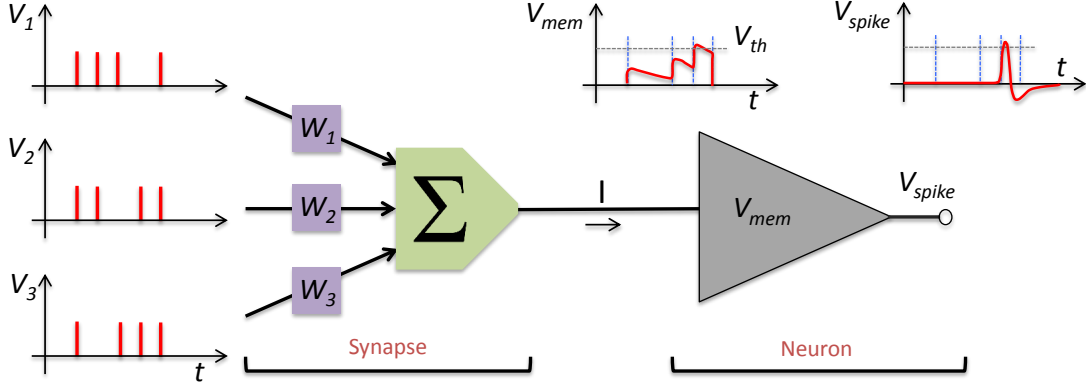


Figure 6.4. Typical SNN dynamics. The input spikes are modulated by the synaptic weights, and the accumulated synaptic current is fed to the neuron. The neuron integrates the current and outputs a spike (fires) once its membrane potential exceeds a threshold.

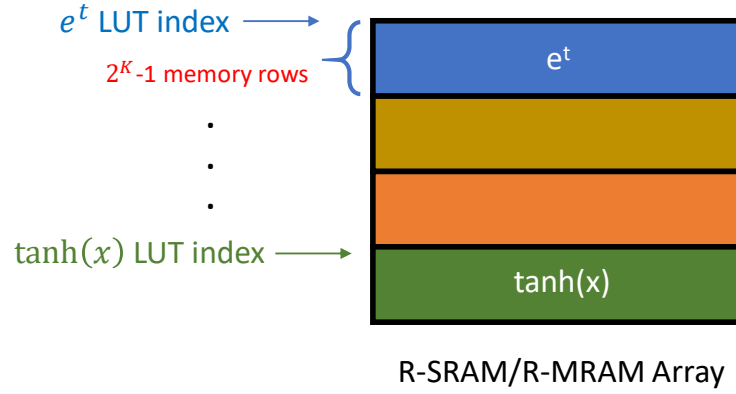


Figure 6.5. Storage of LUTs for various functions within the same ROM-Embedded RAM array. The starting address for each type of LUT is predefined. An offset address (calculated from the input) is added to the starting address to perform the table lookup from the R-SRAM/R-MRAM. The number of memory rows required by each LUT type is predefined based on the desired precision of the transcendental function to be stored.

we obtain improvements in energy consumption at the system level.

R-MRAM An R-MRAM is a memory structure made with conventional STT-MRAM array by embedding a hardware ROM. This allows it to operate in both ROM and RAM mode [115]. As shown in Fig. 6.3, R-MRAM bit cells consist of an additional Bit Line (BL)

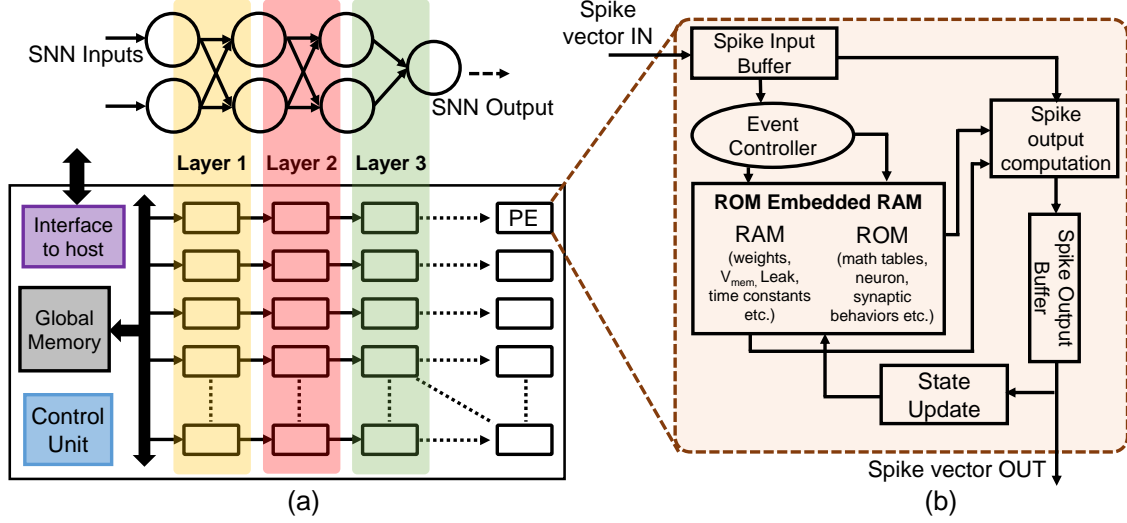


Figure 6.6. Block level diagram of SPARE. (a) Figure shows how a deep neural network is mapped to a 2-D array of PEs connected together. The global memory stores the spiking events at every layer output, and broadcasts them to the input of next layer. (b) Figure zooms into the logical diagram of the PE. It consists of a ROM-embedded RAM to store the state variables along with LUTs of synapse, neuron and synaptic plasticity models, computation core to generate output spikes, input buffers to store incoming spike broadcast, event controller to schedule memory transactions, state updater to update the entries in the memory, and an output buffer to store the output spikes generated.

compared to STT-MRAM. The physical connection of the bit cell (fixed during design time), stores ROM data. Bit cells connected to BL0 store ROM data ‘0’, whereas those connected to BL1 store ROM data ‘1’. Every bit cell can be written/read for RAM operation by electrically connecting BL0 and BL1. However, ROM access and RAM access cannot occur simultaneously. Next we describe the R-MRAM operation for RAM mode and ROM mode of operations.

1. *RAM mode:* During a RAM mode read operation, current from the read-bias generator flows through the pass transistors and the selected bit cell to Select Line (SL) (shown in Fig. 6.3). Consequently, a voltage appears on the positive input of the sense amplifier. The sense amplifier compares the voltage (dependent on the resistance of the selected bit cell) to a reference voltage to output a ‘1’ or ‘0’. For a write operation, EnRAM is asserted to turn ON the pass transistors and the write driver drives both BLs and SL.

2. *ROM mode*: For a ROM read operation, EnRAM is deasserted to turn OFF the pass transistors and the latch is turned ON. If the selected bit cell is connected to BL1 (BL0), BL1 (BL0) gets discharged and ROMOut outputs a ‘1’ (‘0’). Contrary to R-SRAM, the non-volatility of STT-MRAM prevents the RAM data to be lost in R-MRAM during a ROM read operation.

It has been shown in prior studies that the R-MRAM design with an extra BL has no area overhead at array-level. Additionally, this doesn’t impact the performance of the memory as a ROM [115]. Note that during ROM Mode, RAM data is not disturbed due to non-volatility of R-MRAM, thereby simplifying the ROM retrieval process. This results in higher energy benefits of using R-MRAM in SPARE, as we will show later in our simulations.

6.2.2 SNN: Spiking Neural Networks

SNN has emerged as a power-efficient choice for cognitive applications. SNNs are built using bio-plausible neurons and synapses. All information flow is converted into a train of spikes, similar to the information flow in the human brain. Refer to Fig. 6.4. The input spikes V_i are modulated by the synapse weight W_i . At every time-step, V_i is either ‘1’ (spiking event) or ‘0’ (no spike), whereas W_i is a number between -1 and 1, signifying the strength of the connection. The output from all synapses is summed up and fed to the next neuron. The neuron keeps track of its membrane potential (V_{mem}), which gets updated based on the synaptic current. Subsequently, V_{mem} accumulates/decays over several time-steps until it reaches a certain threshold V_{th} , when the neuron emits an output spike (‘1’). This spike is then transmitted to the neurons in the next layer. Depending on the neuron model, V_{mem} dynamics differ in behavior and complexity. During the training phase, the synaptic weights W_i undergo changes to learn the input patterns. Many spike-based learning rules have been proposed, for example, Spike Timing Dependent Plasticity (STDP) [122], Long-Term Potentiation [123] *etc.* The basic idea is to determine the correlation between the input and output neuron spiking activities, to determine the corresponding synapse weight updates. However, once the weights are all trained, the synaptic strengths remain unchanged during the inference phase. These plasticity rules are the basis for unsupervised learning in SNNs.

6.2.3 LUT based storage in R-SRAMs and R-MRAMs

The computations required in the SNN described above rely heavily on transcendental functions and polynomial evaluations. The dynamics of V_{mem} , synaptic current flow, STDP learning, *etc.*, all require solving differential equations with mostly exponential and higher order polynomial evaluations. The only efficient way to compute these functions in hardware is by the use of math tables or LUTs [124]. Taking an example of a typical STDP evaluation in SNNs, we show how the LUTs are structured in R-SRAM/R-MRAM.

1. STDP involves a synaptic weight update, based on the time difference of post- and pre-neurons ($t = t_{post} - t_{pre}$). According to this empirical rule, the change in the synaptic weight is proportional to e^t .
2. Range reduction: t can have an arbitrary value. Thus, t is broken into $N\frac{\log 2}{2^K} + r$, where K is designer's choice that determines the size of LUT, and N is $\lfloor t/\frac{\log 2}{2^K} \rfloor$. Thus the remainder r has a confined range of $|r| \leq \frac{\log 2}{2^{K+1}}$. Thus, the exponential e^t is reduced to $2^{N/2^K} e^r$.
3. Approximation: Due to limited range of r , e^r can be approximated with lower order polynomials (since $e^r = 1 + r + \frac{r^2}{2!} + \dots$).
4. Reconstruction: To evaluate $2^{N/2^K}$, let $N = M2^K + d$, where $M = \lfloor N/2^K \rfloor$ and $d = 0, 1, 2, \dots, 2^K - 1$. Thus $2^{N/2^K} = 2^M 2^{d/2^K}$. Using d as a memory address to the R-SRAM/R-MRAM, the corresponding ROM data (LUT) is fetched, which stores $2^{d/2^K}$. The exponential reduces to $e^t = 2^M \times LUT(d) \times e^r$. The multiplication by 2^M is a simple shift operation in hardware.
5. The exponential e^t is thus used to evaluate the weight update, completing one STDP evaluation.

Other transcendental functions and polynomials can be similarly mapped to LUTs, as described in detail in [124]. Various LUTs are stored within the same array, as shown Fig. 6.5. The starting address of each LUT is pre-defined and is used to perform table lookups.

In the example taken above, when a ‘Fetch LUT’ command is issued, two inputs are provided – the type of LUT (exponential) and the offset (‘d’). The memory address from which the lookup needs to be made is calculated by adding the offset to the LUT index corresponding to the exponential LUT.

6.3 SPARE: SNN Accelerator using ROM-embedded RAMs

6.3.1 SPARE Organization

We propose SPARE, a many-core architecture designed for efficient acceleration of SNNs. As shown in Fig. 7.3(a), it consists of a 2 dimensional PE-array coupled with a global memory and central control unit. A PE can perform all synapse and neuron functionalities required by different types of SNNs. This flexibility is essential as SNN computations typically differ at various levels - neurons, synapses and synaptic weight updates, depending on the application. Layers of an SNN are spatially partitioned across different PEs depending on the network size. The number of neuron state variables and synaptic weights each PE can store is limited by the memory contained within each PE. Based on the network size, number of PEs mapped to each layer are specified.

The SNN computation occurs in time-steps. At each time-step, the neuron firing data is transferred from one layer to the next. Input data spikes (for a given time-step) stored in the global memory are broadcast over the shared bus. Subsequently, the PEs mapping the first layer of the SNN start buffering the data and execute their SNN partition. Once spikes for the first layer have been transmitted, the spikes for the next layer are broadcast, and PEs mapped to second layer start their computations, and so on. All synaptic data is stored locally within each PE. Once the layer-1 PEs finish their execution, their output data (spikes) are written back to the global memory. Subsequently, data from all PEs is written back into the global memory, layer by layer. Consequently, this successive data transfer (neuron data) between global memory and PEs realize a time-step of SNN computation. It’s worth noting that only neuron data movements occur between PEs and global memory, whereas the synapse data is locally read from the PE’s RAM. This reduces the data movements in SPARE compared to a von-Neumann machine which would involve moving both neuron

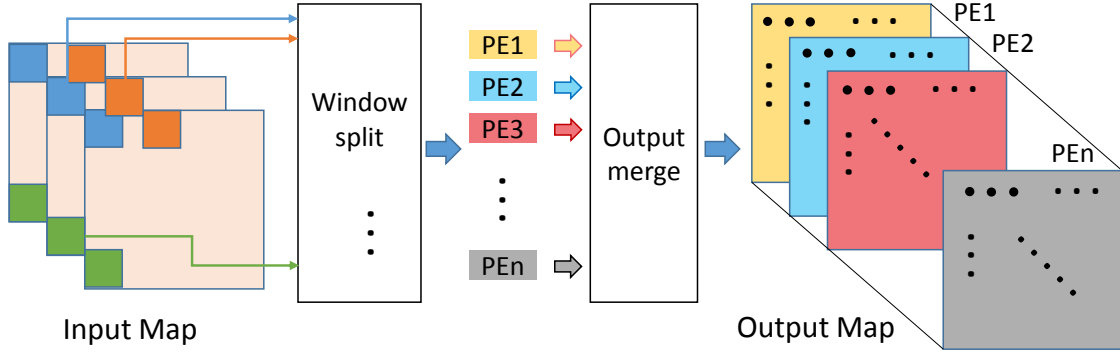


Figure 6.7. Mapping of CNNs in SPARE: The input map is window-split based on the kernel size of the particular layer. These are then broadcast to all PEs mapped to that layer. Each PE stores different kernels, and process the data in parallel as they receive the inputs in a window split-manner. Each PE computes part of the output feature map, highlighted through color coding of PEs in figure. The output is rearranged and stored back to the global memory unit.

and synapse data between the global memory and the computation core. Additionally, this reduction is extremely significant as typical SNNs have $1000\times$ more synapses than neurons [125].

We extend this approach to map convolutional neural networks (CNNs) using SPARE. CNNs have been shown effective for image classification tasks, achieving state-of-the-art accuracies, occasionally surpassing human performance [52], [53]. The standard architecture consists of alternate convolutional (c-) and spatial-pooling (s-) layers, followed by a final fully-connected (fc-) layer. Each convolutional layer hierarchically extracts complex features from the input image. This is done by using shared weight kernels that perform a convolution operation on the input image. The output of one convolutional layer becomes the input of the next. Thus, the kernels in the first convolutional layer learn low-level features, for example, edges and corners, while in deeper layers, they learn high-level features, using these low-level features as inputs. A spatial-pooling layer is added in between two convolutional layers to reduce the dimensions of the convolutional feature maps. This layer maintains the depth of the input map, however reduces the spatial dimensions. Finally, a fully-connected layer is used to determine the output class of the input image. Fig. 6.7 shows how the

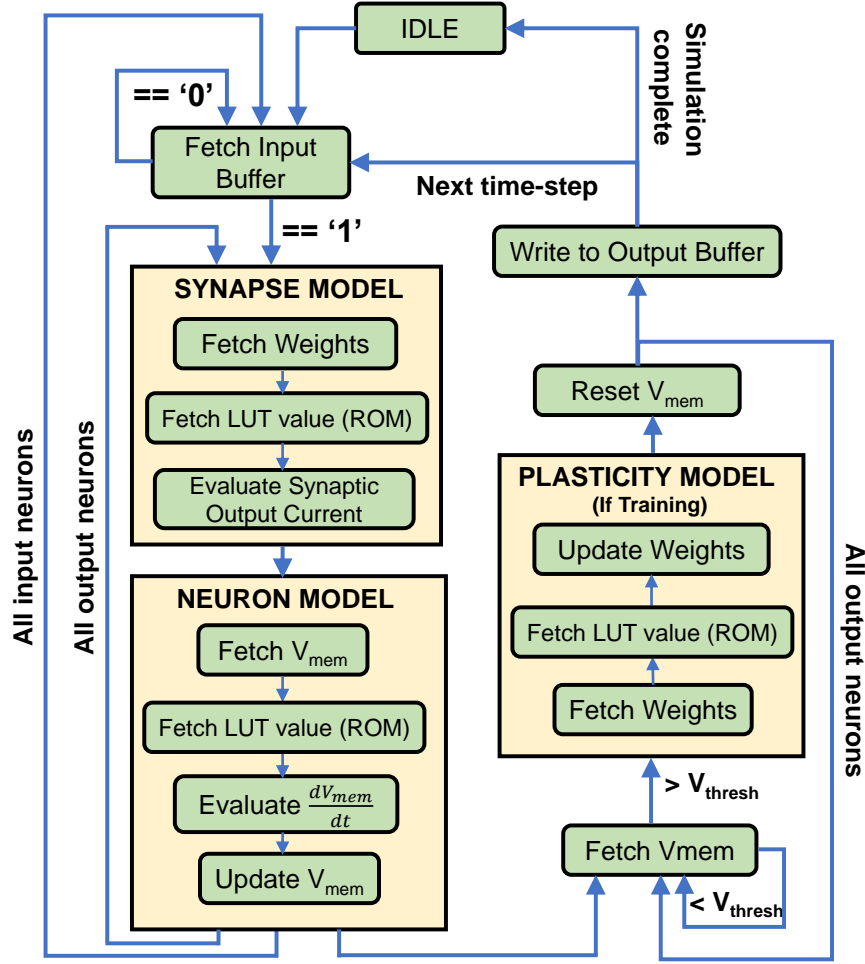


Figure 6.8. Logical flow diagram of the event controller, describing SNN computations performed in the PE. The subsequent computation is subdivided into three main blocks. 1) Synapse model block: computes output synaptic current. 2) Neuron model block: keeps track of the membrane potential of output neurons. 3) Plasticity model block: updates synaptic weights during the training phase. This block is skipped during the inference phase.

convolutional layer can be mapped to SPARE. The input map is split using a small window that strides throughout the image. The window size is governed by the kernel size of that layer. Input spikes are broadcast to the PEs in this window-split manner (instead of pixel-by-pixel manner), where each PE stores a different kernel of that layer. Thus, each PE computes part of the output feature map, which is then merged and stored in the global memory unit, as shown in the figure. Layer parameters (for example, stride, kernel size and number of output maps) are programmed into the global control unit to implement this ‘window-split

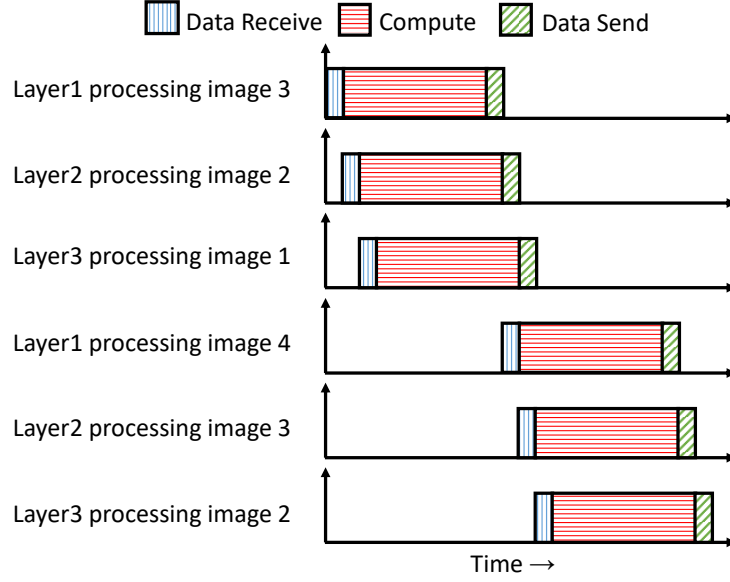


Figure 6.9. Timing diagram illustrating the inter-layer pipelining in SPARE. As soon as the PEs receive and buffer the input data, they start processing. Meanwhile, data for PEs mapped to subsequent layers is transmitted. Since the data transfer time is small compared to the computation time within the PE, all PEs process data in parallel.

input broadcast’ and ‘output merge’ in the global memory. Note that the s- and fc- layers can be configured as c- layers, with appropriate parameters. For s- layer, the parameters are: stride = 2, kernel size = 2×2 , number of output maps = number of input maps. Whereas for an fc- layer, stride = 0, kernel size = input feature size, number of output maps = number of output neurons. Thus, the proposed architecture is a generalized programmable architecture that maps convolutional, spatial pooling as well as fully-connected layers.

6.3.2 Inter-layer pipelining

SPARE enables a pipelined execution of layers in an SNN to exploit the available inter-layer data parallelism. Layers of SNN are mapped across the 2-dimensional PE array. Hence, while layer-2 PEs are computing the n^{th} input image, layer-1 PEs compute the $(n+1)^{\text{th}}$ input image and so on. Data communication between layers of SNN are achieved by scatter and gather operations initiated by the SPARE control unit (see Fig. 7.3(a)) to move data between global memory and PE-array. SPARE control unit stores the mapping information between

SNN layers and PEs. A gather operation for a layer collects the output data computed by the PEs mapped to the specific layer and stores it in the global memory. Scatter operation for a layer sends the input data to the required PEs. It is important to note that data communication in SNNs is of feed-forward nature where PEs mapped to layer- n will only send data to PEs mapped to the subsequent layer- $n+1$ and so on. Hence, we do not support a dedicated on-chip network for all PE-to-PE communication due to the associated area and power overheads. Our “in-memory” nature of computing results into PEs spending more time in computation (within PE) rather than sending and receiving data from global memory. Hence, our inter-layer communication based on a shared resource (global memory) doesn’t lead to performance issues due to the natural pipelining obtained as shown in Fig 6.9.

6.3.3 Processing Element (PE)

As shown in Fig. 7.3(b), PE contains a computing core to perform SNN computations and a memory unit to store the neuron-synapse models, state variables and LUTs. The memory unit (RAM and ROM) and the computation core within the PE localize most of the data movements required for computing the output neurons (mapped to the PE), thereby enabling “in-memory processing”. While RAM houses all the synaptic weights and state variables required for the output neuron computations, the ROM stores the LUTs required for modeling synapse, neuron and synaptic weight update computations. Consequently, the higher storage density enabled by ROM-embedded RAMs (smaller memory size) and the resulting reduction in data movements increases the computation efficiency and reduces overall energy consumption. The computational flow in a PE and a step-by-step procedure for typical SNN computation is illustrated in Fig. 6.8. It consists of three main blocks: 1) Synapse model, 2) Neuron model and 3) Plasticity model. The event controller checks the head of the input spike buffer, and both the Synapse and Neuron blocks are skipped if the input is ‘0’, thereby leveraging the benefits of event-driven computing in SNNs to achieve energy-efficiency. Similarly, the Plasticity block is skipped if the V_{mem} is less than the threshold (no synaptic weight update). PEs are modeled as extended finite-state machines. As soon as the PE receives the broadcast of spikes corresponding to its layer tag, it starts

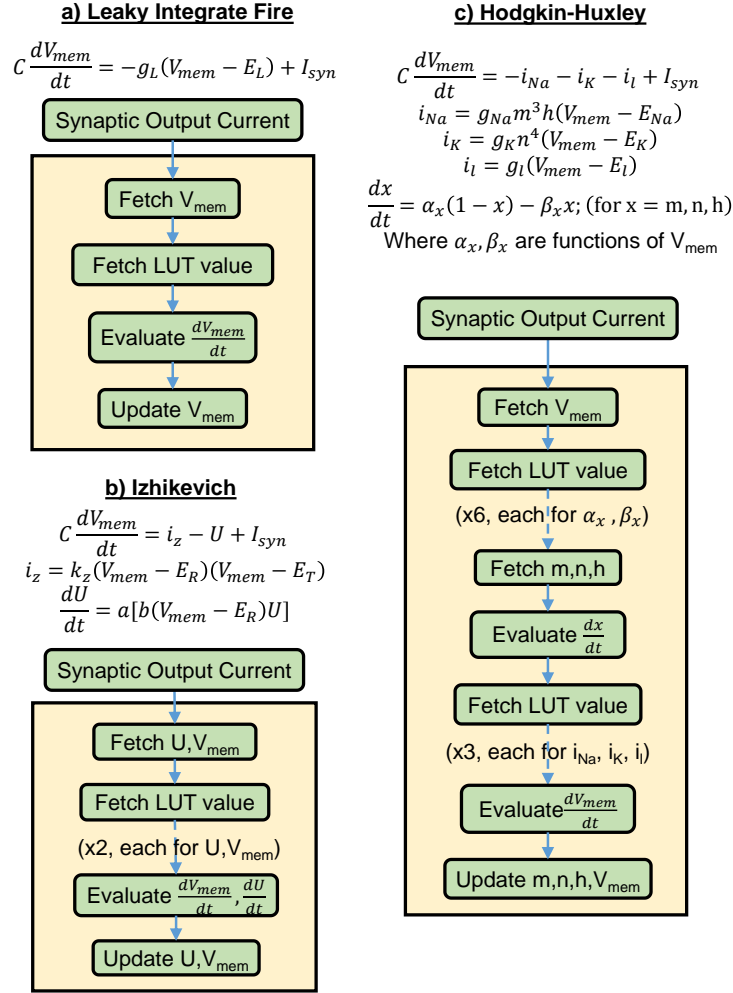


Figure 6.10. Differential equations describing the dynamics of neurons and an LUT based approach to implement them in SPARE. (a) Leaky-integrate-fire neuron (b) Izhikevich neuron (c) Hodgkin-Huxley neuron

computing. Thus, effectively all PEs run in parallel, exploiting data-parallelism and inter-layer pipelining. Since the input spikes are broadcast to all PEs, each PE performs the SNN computation corresponding to the neuron and synapses it is mapped to. Since SPARE localizes data-movement through in-memory computing, the same memory storage unit also contains the LUTs used in SNN computations allowing a simple and compact PE design.

Design	Energy (pJ)			Latency (ns)			Leakage (mW)
	RAM Read	RAM Write	ROM Read	RAM Read	RAM Write	ROM Read	
SRAM	38.42	33.39	38.42	0.53	0.53	0.53	74.89
R-SRAM	16.99	14.48	62.94*	0.418	0.418	1.254*	40.92
STT-MRAM	35.48	146.31	35.48	1.18	10.34	1.18	0.72
R-MRAM	17.93	73.37	17.93	1.16	10.32	1.16	0.48

Figure 6.11. Energy and latency for read-write accesses from all designs considered – SRAM, R-SRAM, STT-MRAM, and R-MRAM. (* ROM Read for R-SRAM includes additional overhead of buffering RAM, retrieving ROM data and storing back the buffered RAM data, as described in Section 6.2.1).

Parameter	Value
Frequency	1 GHz
Technology node	45 nm
Memory unit	32KB ROM-embedded RAM
Buffer depth	32
Synapse weight precision	8-bit
Neuron V_{mem} precision	8-bit
Data width	32-bit
ALU registers	32-bit fixed point
PE core static power	1.311 mW

Figure 6.12. uArchitecture design parameters used for simulations.

6.3.4 Modeling complex neuro-synaptic functionality

Most neuron-synapse models have complicated differential equations, and heavily use higher order polynomials and transcendental functions. This makes them highly suitable for an LUT based storage in ROM-embedded RAMs. Thus, our PE incorporates any model needed by the SNN application without much area overhead. To illustrate this, dynamics of three different neuron models - LIF [126], Izhikevich [127], and Hodgkin-Huxley (HH) [128] are shown in Fig. 6.10. Each model can be implemented in SPARE, by modifying the ‘neuron model’ block in the state diagram (see Fig. 6.8), with corresponding alterations. As shown in Fig. 6.10(c), the HH model is described by 7 differential equations, with 4 state

Benchmark	Network Configuration	# PEs	Memory (MB)	RAM content	ROM content
MNIST-1	784 x 400	16	0.5	Synaptic weights State variables (V_{mem}) Spiketimes	Synapse model: I_{SYN} as a function of weights w_{ij} Neuron model (LIF): $\frac{dV_{mem}}{dt}$ as a function of V_{mem} Neuron model (HH): α_x, β_x as a function of V_{mem} i_{Na}, i_K, i_l as a function of V_{mem}, x ($x=m,n,h$) Plasticity block: exponential LUT for STDP learning
MNIST-2	784 x 1600	50	1.56		
MNIST-3	784 x 6400	200	6.25		
MNIST-4	784 x 1200 x 1200 x 10	100	3.125		
CIFAR10	32x32x3-24c5-2s-80c5-2s-10o	220	6.875	Kernel weights State variables (V_{mem})	

Figure 6.13. SNN benchmarks used in SPARE evaluation [118], [129]. The figure tabulates the number of PEs required, memory requirement, and the RAM/ROM content for each benchmark and neuron model.

variables - V_{mem}, m, n and h . Firstly, we need 4 RAM fetches to read the state variables. To update m, n , and h , we need a total of 6 ROM fetches, each for $\alpha_{m,n,h}, \beta_{m,n,h}$. Next, we need to evaluate higher order polynomials (also stored in LUTs) in order to calculate $i_{Na,K,l}$. Thus, a total of 9 ROM fetches, 4 RAM fetches and 4 RAM updates per spike per time-step are required for HH model, in contrast to 1 ROM fetch, 1 RAM fetch and 1 RAM update in case of a simple LIF model. However, the overall data flow diagram remains unchanged. A similar approach can be used to implement various synapse and plasticity models, by modifying the synapse and plasticity blocks, respectively. As the models become more complex, more LUTs are required to store multiple polynomial functions and math tables. Moreover, the number of ROM and RAM fetches also increase. SPARE addresses both these issues since the ROM-embedded RAM primitive allows *extra ROM* for LUT storage, thereby allowing a compact memory unit. Data-localization in SPARE along with the compact memory storage unit enables a lower energy/latency per ROM/RAM access.

6.4 Experimental Methodology

PE was modeled at the Register Transfer Level (RTL) in Verilog and synthesized to IBM 45nm technology library using Synopsys Design Compiler to estimate the power and area consumptions. R-SRAM and R-MRAM (memory units) were modeled using Cacti [76] and NVSim [130], respectively, for the corresponding RAM sizes at 45nm technology node. Subsequently, we account for the modified ROM access cycles and peripheral circuits described in Sec. 6.2.1. Fig. 6.11 summarizes the RAM/ROM read-write energy and latency obtained

from simulations for SRAM, R-SRAM, STT-MRAM and R-MRAMs. Cycle-accurate RTL simulations were performed to get estimates of memory (RAM, ROM) access traces and subsequently, the overall energy consumption per classification. Fig. 6.12 summarizes various μ -architecture parameters used for the simulations.

We analyze the energy, performance and area benefits of SPARE on MNIST dataset [102] and CIFAR-10 dataset [103]. For an apples-to-apples comparison, we use a similar architecture built with PEs comprised of typical RAM and STT-MRAM as our baselines (without ROM-Embedded RAM capability). Additionally, to demonstrate system scalability, we benchmark SPARE with various network sizes of different scales, varying from 1184 to 36602 neurons. Fig. 6.13 tabulates the benchmarks chosen [118], [129], [131], and the number of PEs required in each case. Note that benchmarks ‘MNIST-1,2,3’ are typical two-layer SNNs, that can be trained using STDP learning [129]. The input layer has 784 neurons, each corresponding to an input pixel in the image. The output layer has 400, 1600 and 6400 neurons for benchmark ‘MNIST-1,2 and 3’ respectively. For deep spiking networks beyond two-layers, there hasn’t been any successful attempt to generalize a training algorithm in the spiking domain. However, [118], [131] show that off-line training of the network using DNN techniques (standard back-propagation algorithm) and converting the trained network to an SNN does not incur significant performance degradation. Thus, to evaluate SPARE on deep networks, we use benchmarks ‘MNIST-4’ and ‘CIFAR10’, in the inference phase. ‘MNIST-4’ is a deep multi-layered, fully connected SNN converted from a trained DNN [118]. It consists of an input layer of 784 neurons, followed by two hidden layers with 1200 neurons each, and finally an output layer of 10 neurons. Benchmark ‘CIFAR10’, on the other hand, is a deep convolutional neural network converted from a trained CNN (32x32x3-24c5-2s-80c5-2s-10o). The CNN has two convolution (c-) and two spatial-pool (s-) layers arranged alternately, followed by a fully-connected (fc-) output layer. The dimension of input image is 32x32x3. The first c- layer consists of 24 kernels of size 5x5x3. The following s- layer has kernels with size 2x2. The second c- layer has 80 kernels of size 5x5x24, followed by another s- layer with kernel size 2x2. The final layer has 10 neurons, fully connected to the previous layer. In all our simulations, we use the LIF neuron model along with the exponential STDP based plasticity for the training phase.

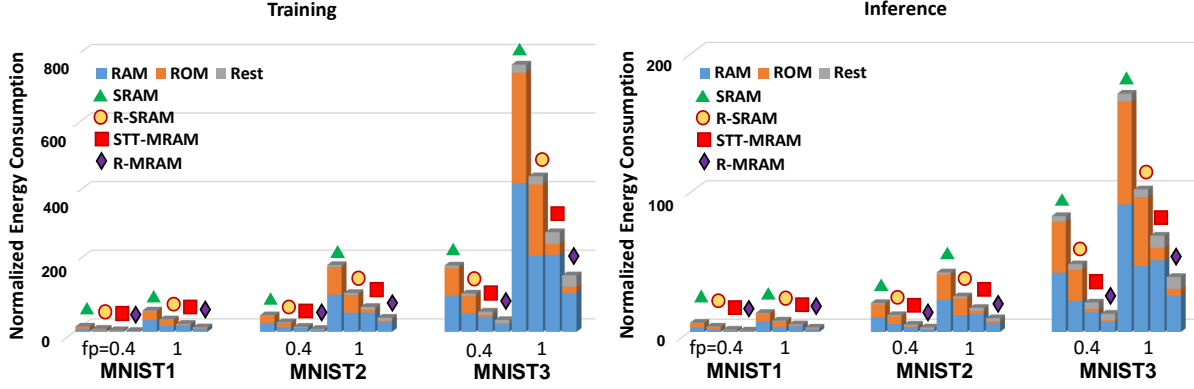


Figure 6.14. Normalized energy consumption for a) Training phase and b) Inference phase, for benchmarks ‘MNIST1-3’. The simulations are performed for max firing rate $fp = 0.4$ and 1 . The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.

Input spike trains were generated from the input image pixels based on the rate-coding approach used in [132]. Each image is split-up into several time-steps, each conveying the input firing activity. We analyze the benefits of SPARE towards leveraging SNN data sparsity (event-drivenness) by analyzing each SNN on different input maximum firing rates, $fp = 0.4$ and $fp = 1$ [131]. Kindly note that we use the SNN size and dataset statistics only for exploring system scalability and benefits of in-memory computing for training and testing SNNs. Mapping of these networks to the proposed architecture doesn’t lead to any degradation in the classification accuracy. Readers are referred to [118], [129], [131] to explore the classification accuracy achieved in the benchmarks used.

6.5 Results

6.5.1 Energy

A common base reference was used to normalize all energy numbers obtained through simulations such that the minimum energy consumption bar (Inference of MNIST-1 for R-MRAM with $fp=0.4$) represents 1. All other energy bars in Fig. 6.14, Fig. 6.15, and Fig. 6.17 are normalized to this value. Fig. 6.14 shows the energy consumption for benchmarks ‘MNIST-1,2,3’, both for training and inference phases. Each bar shows the total energy,

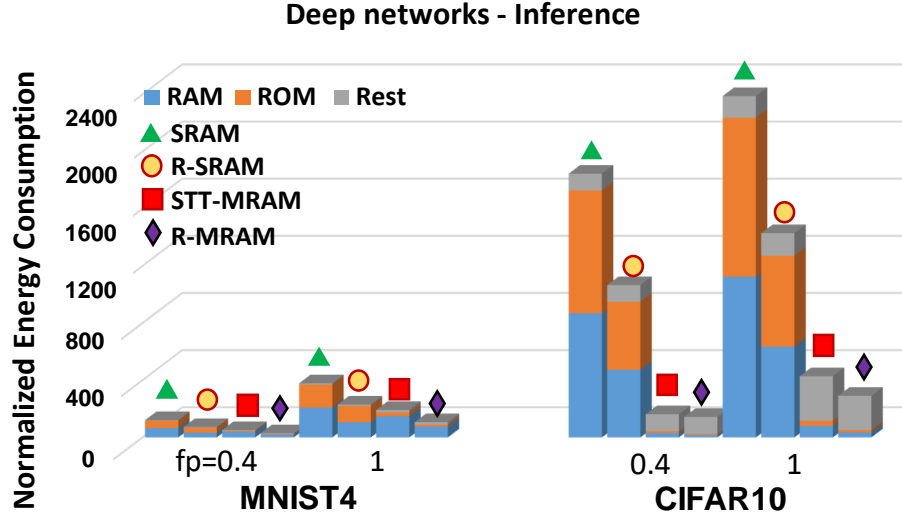


Figure 6.15. Normalized energy consumption for benchmarks ‘MNIST4’ and ‘CIFAR10’. The simulations are performed for max firing rate $fp = 0.4$ and 1. The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.

which is further split into three sub-components 1. RAM (access + leakage) 2. ROM (access + leakage) and 3. Rest (Core - buffer, control, compute). The following observations can be drawn from Fig. 6.14. 1) It can be seen that an increase in the maximum firing rate (fp) results in increased overall energy consumption across all datasets. This is because a higher firing rate results in increased number of spikes. Consequently, this increases the number of RAM/ROM accesses, thereby decreasing the benefits from event-driven computing in SPARE. This also increases the overall computations as more synapses will be accumulated over the output neurons. This underscores the effectiveness of SPARE in drawing benefits from the data sparsity in SNNs. 2) The total energy consumption in the inference phase is lower compared to the training phase because the Plasticity block (refer Fig. 6.8) is skipped during the inference phase, as described in Sec 6.3.3. 3) The energy consumption with STT-MRAM technology is more than $\sim 2\times$ less, compared to CMOS based memory technology.

This was expected since STT-MRAM is a NVM, thus, leakage due to memory is close to 0. Although writing into STT-MRAM is expensive compared to CMOS, the near-zero

leakage is a dominant factor in reducing the energy consumption. 4) Using R-SRAM and R-MRAM as the memory units in the PE, we obtain, $1.71\times$, and $1.76\times$ reduction in energy consumption on an average, compared to CMOS SRAM and STT-MRAM, respectively. This is a direct consequence of increased storage density (or, smaller area for iso-bytes) provided by ROM-embedded RAMs. A smaller memory reduces the access energy and the leakage, thereby leading to energy benefits. However, note that for iso-area, higher storage density (through ROM-embedded RAMs) allows bigger on-PE storage, eliminating data movements required from external memory (in case of typical RAM). This leads to energy benefits. Note that we have used iso-storage PEs in our simulations to evaluate the energy benefits. 5) The energy improvement in STT-MRAM technology is greater compared to CMOS due to a simpler ROM retrieval process in R-MRAMs compared to R-SRAM (refer Sec. 6.2.1). R-SRAMs require additional steps in buffering the RAM data, for each ROM access, which is not required in R-MRAMs.

Moving to deeper networks, Fig. 6.15 shows the energy consumption for deep networks, illustrating the scalability of SPARE towards executing SNN workloads. A few additional observations can be inferred: 1) Benchmark ‘MNIST4’ being a deeper extension of ‘MNIST1-3’, obtains similar improvements of $1.65\times$, and $1.77\times$ reduction in energy consumption for CMOS and STT-MRAM technologies, respectively. 2) For a deep convolutional network (‘CIFAR10’), the improvements are $1.70\times$ and $1.31\times$, for CMOS and STT-MRAM, respectively. CNNs are more compute-intensive compared to memory-intensive fully connected networks [133]. Thus, more energy is spent in computations, compared to the memory transactions. Thus, the energy consumed by the core and the memory leakage energy are significant. For the CMOS case in benchmark ‘CIFAR10’, the memory leakage overwhelms the core energy consumption (see Fig. 6.15), whereas in STT-MRAM, the core energy consumption overwhelms the memory energy (no leakage!). Due to this reason, the improvement of using R-MRAMs is suppressed in CNNs. Comparing only the memory energy consumption (RAM+ROM), we still obtain $\sim 2\times$ improvement for R-MRAMs, however, the core energy being dominant reduces overall benefits. Note that using the STT-MRAM technology itself decreases the energy consumption by an order of magnitude compared to CMOS. Thus, we conclude that using R-SRAMs over typical SRAMs as compute units lead

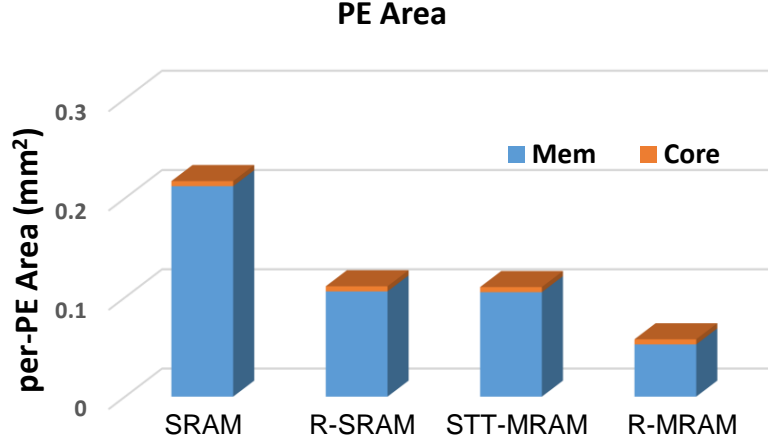


Figure 6.16. per-PE area with SRAM, R-SRAM, STT-MRAM and R-MRAM as memory units (for iso-storage).

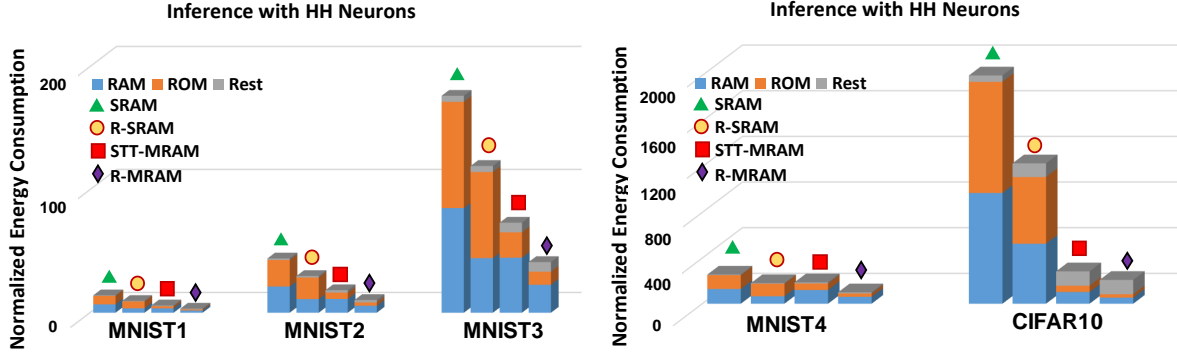


Figure 6.17. Normalized energy consumption for using Hodgkin-Huxley neuron models on SNN benchmarks. The simulations are performed for max firing rate $fp = 0.4$. The energy bars are further split into RAM (read/write energy + leakage), ROM (read energy + leakage) and Rest (core energy). The energy values are normalized to the common base reference.

to $\sim 1.7\times$ improvement in energy for both fully-connected networks and convolutional networks. Whereas, using R-MRAMs over typical STT-MRAMs lead to $\sim 1.75\times$ improvement for fully-connected networks, and $\sim 1.3\times$ for CNNs.

6.5.2 Area

By using R-SRAMs and R-MRAMs in PEs, $1.95\times$ and $1.91\times$ area benefits are achieved on a per-PE basis, for R-SRAM and R-MRAM, respectively, shown in Fig. 6.16. This is because ROM-embedded RAM effectively provides extra ROM with no area overhead. Moreover, the PE area is dominated by the memory unit, since the core (buffers, controller and computation core) consumes a small portion of the total area. The area consumption of the shared bus and global memory are insignificant with respect to the total PE area (hundreds of PEs used in benchmarks - see Fig. 6.13). This translates to SPARE being more area-efficient compared to a normal-RAM based system.

6.5.3 Performance

In the previous section, we observed a reduction in PE area by a factor of $1.91 - 1.95\times$ by using R-SRAMs/R-MRAMs, due to higher storage density provided by ROM-embedded RAMs. For a given chip area (iso-area), we can fit about twice as many PEs that use R-SRAM and R-MRAM compared to typical SRAMs and STT-MRAMs, respectively, by using smaller memory sizes. Computations (neurons in a layer) can be split between more PEs, translating to $1.91 - 1.95\times$ performance benefits. Note that we assume a ROM:RAM ratio of 1:1. This is reasonable for SNN computations due to extensive LUT demands arising from various math function requirements. However, if the designer wishes to decrease the ratio (at the cost of lower precision of LUTs and lower flexibility with respect to neuro-synaptic functionalities), the performance improvement would be smaller, as the ratio decreases.

6.5.4 Complex neuro-synaptic models

We expect to achieve higher benefits in mapping more complicated neuro-synaptic models, due to increased LUT storage demands and ROM accesses per classification. In literature, usage of complicated models, such as the Hodgkin-Huxley (HH) and Izhikevich neuron models, is limited to biological experiments, and no references report a decent classification accuracy in using such models in SNN classification tasks. However, in order to evaluate

SPARE for more complex models, we estimate the energy benefits of using HH neuron model for the same benchmarks used before. Note that the level of complexity of the differential equations increases from LIF to Izhikevich to HH, as described earlier in Section 6.3.4. For LIF, 1 ROM fetch, 1 RAM write, and 1 RAM read is required per computation. For Izhikevich, 2 ROM fetches, 2 RAM writes, and 2 RAM reads are required. While for HH, 9 ROM fetches, 4 RAM writes, and 4 RAM reads are needed. Thus, it is trivial that the energy consumption would increase as we go from LIF to Izhikevich to HH. To avoid clutter, we only compare the two extreme cases (LIF and HH), to evaluate SPARE with complex neuron models. Fig. 6.17 shows the normalized energy consumption for the inference phase for MNIST and CIFAR10 datasets, using the HH neuron model at max firing rate $fp = 0.4$. Note that these are only projected values showing the energy profiles in using HH neurons for SNN workloads. The following observations can be inferred: 1) The energy consumption is higher, as compared to the LIF neuron implementation, throughout all datasets. Moreover, energy spent in ROM accesses is higher than RAM accesses. This is a direct consequence of additional RAM and ROM fetches (9 ROM fetches, 4 RAM fetches, 4 RAM updates per spike per timestep) involved in solving complex differential equations for HH neurons. 2) For fully connected networks, we obtain $1.45\times$ and $1.84\times$ reduction in energy consumption on an average, for R-SRAMs and R-MRAMs compared to CMOS SRAM and STT-MRAM, respectively. While for convolutional networks, we obtain $1.67\times$ and $1.4\times$ reduction. Note that the corresponding improvements in energy are higher for R-MRAM technology, but lower for the R-SRAM technology, compared to the LIF neuron case (Sec. 6.5.1). This is due to the fact that R-SRAMs have additional overhead in ROM retrieval process, as described earlier. Since HH neurons involve lots of ROM accesses, this overhead leads to a reduction in energy improvements. While for R-MRAMs, since the overhead is minimal, increased ROM accesses leads to higher energy benefits.

6.6 Conclusion

We presented SPARE, an architecture utilizing the ‘in-memory processing’ abilities of ROM-embedded RAM to enable efficient acceleration of SNNs. Each processing unit in

SPARE does event-driven processing in order to leverage the benefits from input data sparsity in SNNs. We analyzed trade-offs of using CMOS based R-SRAMs and STT-MRAM based R-MRAMs as memory units in SPARE for different types of networks. Our experiments on various SNN benchmarks for image classification applications reveal that R-MRAMs are suitable for mapping fully-connected networks compared to typical STT-MRAM arrays with $\sim 1.75\times$ lower energy, while R-SRAMs are suitable for mapping CNNs compared to typical SRAM arrays with $\sim 1.7\times$ lower energy. R-SRAM and R-MRAM achieve $\sim 1.9\times$ reduction in area for iso-storage. Moreover, for iso-area, R-SRAMs and R-MRAMs can achieve $\sim 1.9\times$ improvement in performance, given required data parallelism (neurons in a layer) is available. SPARE also provides the necessary programmability to execute a variety of synapse, neuron and plasticity models thereby, enabling designers to deploy SNNs based on the application requirements. SPARE thus underscores the applicability of ROM-embedded RAM based in-memory hardware primitives in efficient cognitive computing.

7. A 65-NM DIGITAL COMPUTE-IN-MEMORY MACRO ENABLING SPIKE-BASED SEQUENTIAL LEARNING IN 10T SRAM ARRAY

7.1 Introduction

Spiking Neural Networks (SNNs) aim to harness the inherent energy-efficiency arising from highly sparse and event-driven spike-based information processing [134]. SNN algorithms continue to develop rapidly, achieving image classification accuracies close to state-of-the-art [135]. The SNN approach uses binary inputs and outputs (1—spike, 0—no spike) over several timesteps, unlike traditional ANNs. More importantly, the neuron membrane potential plays a key role which defines each neuron’s current state, thereby allowing SNNs to process dynamical (temporal) aspects in the data. This makes them suited for sequential learning tasks, avoiding the complexity of recurrent neural networks, such as Long Short-Term Memory (LSTM). Fig. 7.1 illustrates the processing of sequential inputs in both LSTM and SNN. In LSTM, the hidden state (h, C) stores information about all past inputs the network has seen before [136]. To enable this, the previous hidden state is fed back as input through a recurrent connection along with the current input. Whereas, in SNNs the inherent recurrence in membrane potential acts as a memory to store information about past inputs. Each LSTM layer has $4(mn + n^2)$ parameters, compared to mn parameters in an SNN layer, where m and n are input and output dimensions, respectively.

However, processing of membrane potential over several timesteps incurs additional memory-access bottlenecks, specific to SNNs [137]. The main challenges in current SNN hardware accelerators are: (1) Significant energy consumption due to data movements from weight and membrane potential SRAMs to compute units. (2) They support limited SNN functionality due to the adoption of custom neuron circuitry, which are power and area expensive, making them restricted to simpler tasks. These are illustrated in Fig. 7.2.

To overcome these challenges, we propose IMPULSE [138]: (1) an SRAM-based CIM macro which integrates all instructions required for SNN inference such as accumulate, thresholding, spike-check and reset, within the fused W_{MEM} and V_{MEM} memory, thereby

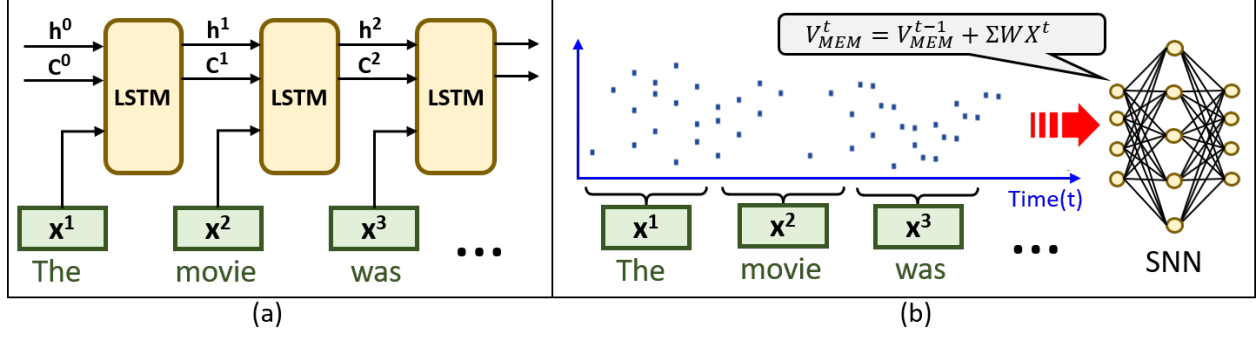


Figure 7.1. (a) LSTM having hidden state (h^t , C^t) for processing sequential tasks. (b) Intrinsic temporal dynamics of neuron membrane potentials (V_{MEM}^t) in SNNs for processing sequential tasks.

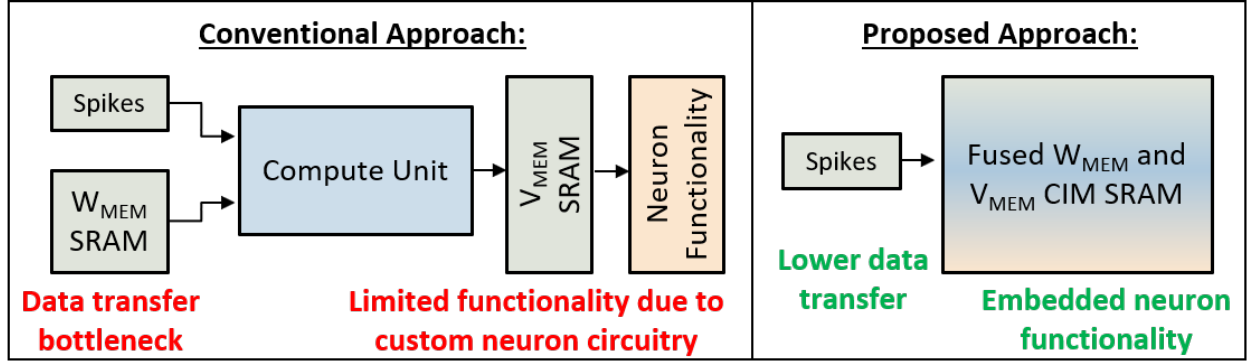


Figure 7.2. Limitations of current digital SNN hardware accelerators and our proposed approach of fused weight and membrane potential CIM SRAM.

reducing on-chip SRAM accesses. (2) Support for multiple types of neurons through the same in-memory instructions integrate-fire (IF), leaky-IF (LIF), and residual membrane potential (RMP) [139] neurons. (3) A staggered data mapping and reconfigurable column peripherals for maintaining different bit-precision requirements of W_{MEM} and V_{MEM} , while allowing full utilization of the array and column peripherals.

7.2 IMPULSE: Structure and Operation

We use a differential 10T-SRAM cell with decoupled read-write port. The triple-row decoder can take three addresses and enables two RWLs and one WWL simultaneously. During the CIM operations, the RBL gives NOR/OR, while RBLB gives NAND/AND, of

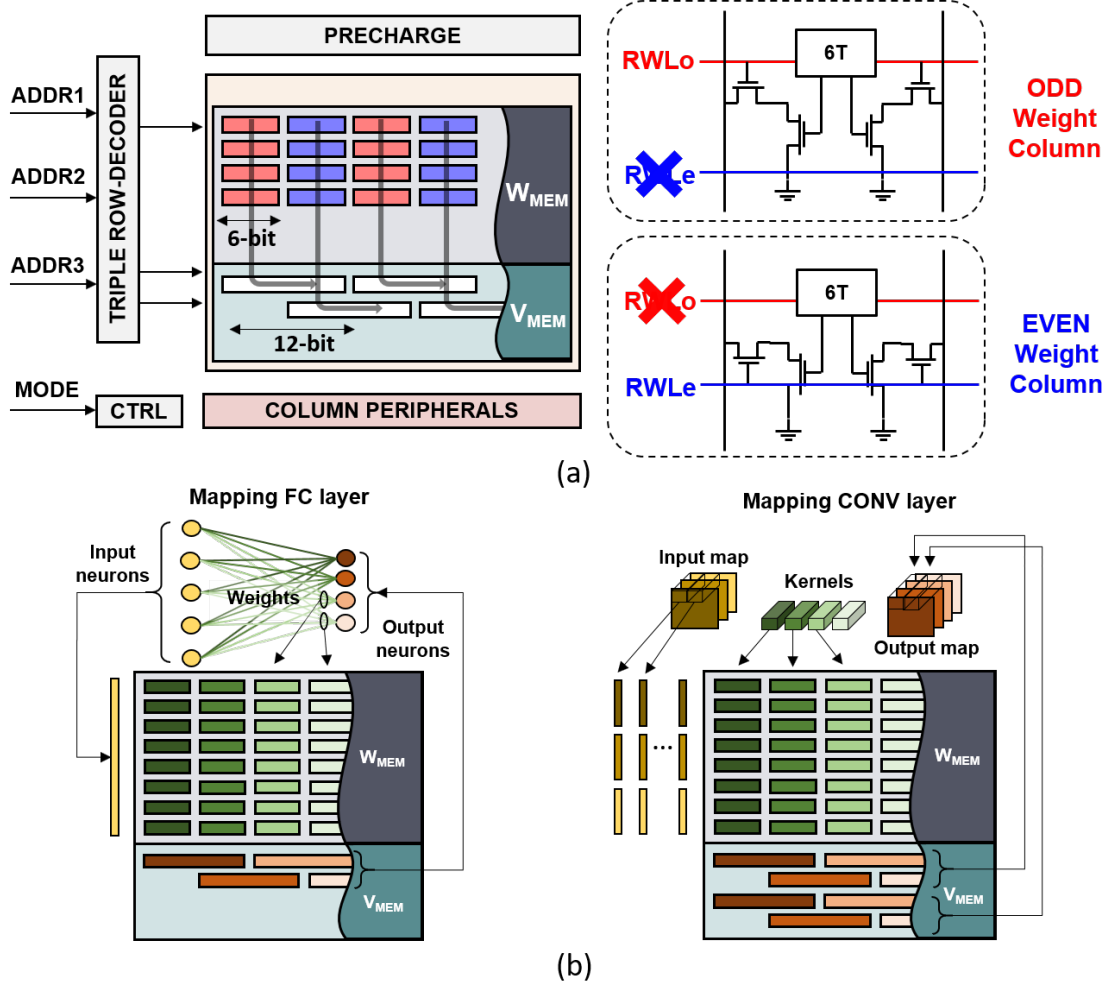


Figure 7.3. (a) Organization of the fused W_{MEM}/V_{MEM} 10T-SRAM macro. (b) Mapping of FC and Conv layers on the proposed macro.

the data from the two enabled RWL rows. Fig. 7.3(a) shows the overall organization of the macro. Each of the 128 rows in the W_{MEM} subarray corresponds to an input neuron, storing twelve 6-bit signed weights. Each row has two read wordlines (RWLo/RWLe) and the weights are interleaved, such that the first six bits are on RWLo, next six on RWLe, and so on. In each cycle, either of RWLo or RWLe are enabled. The V_{MEM} subarray contains 32 rows with single RWL, each row storing six signed values. The V_{MEM} corresponding to odd and even weight columns are stored in different rows with a staggered alignment. This mapping technique compactly handles the different precision requirements for weights (6-bit) and V_{MEM} (11-bit), with full utilization of all column peripherals in each odd/even

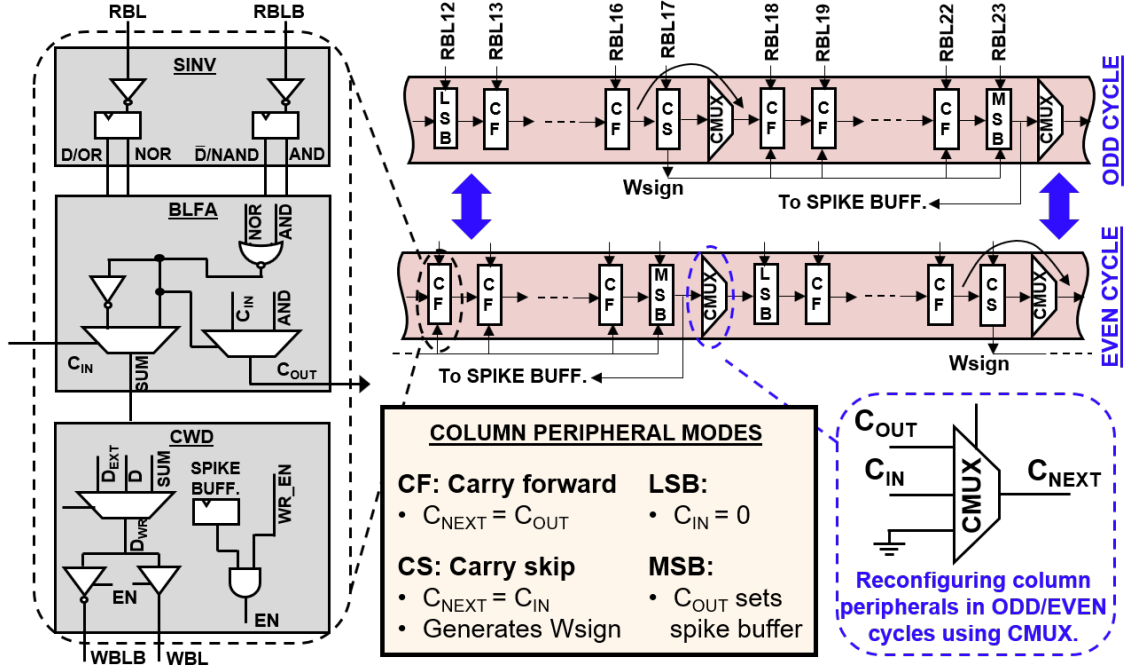


Figure 7.4. Detailed description of the reconfigurable column peripherals.

cycle. The two subarrays are fused through common bitlines. Fig. 7.3(b) illustrates how fully-connected (FC) and convolutional (Conv) layers can be mapped to the macro.

7.2.1 Reconfigurable Column Peripherals

Fig. 7.4 describes the column peripherals in detail. Each set of bitlines (RBL,RBLB,WBL,WBLB) connects to a column peripheral circuitry. The data on RBL/RBLB is sensed and latched using the sensing inverters (SINV). The bitwise logic full adder (BLFA) is designed to generate SUM and COUT using these bitwise signals. SUM is fed to the conditional write-driver (CWD), to be written into the enabled WWL destination row, while COUT is forwarded to the adjacent column peripheral for the accumulate operation, forming a ripple-carry adder.

The modular design of the adder using BLFAs from each column peripheral allows reconfigurability during odd/even cycles. To account for the staggered data mapping, each column peripheral can be reconfigured in carry forward (CF), carry skip (CS), LSB and MSB modes, with the help of Carry-MUXes (CMUX), as shown in Fig. 7.4. For example, during odd cycle, Col[0-11] form one adder, Col[12-23] forms another, and so on. Whereas

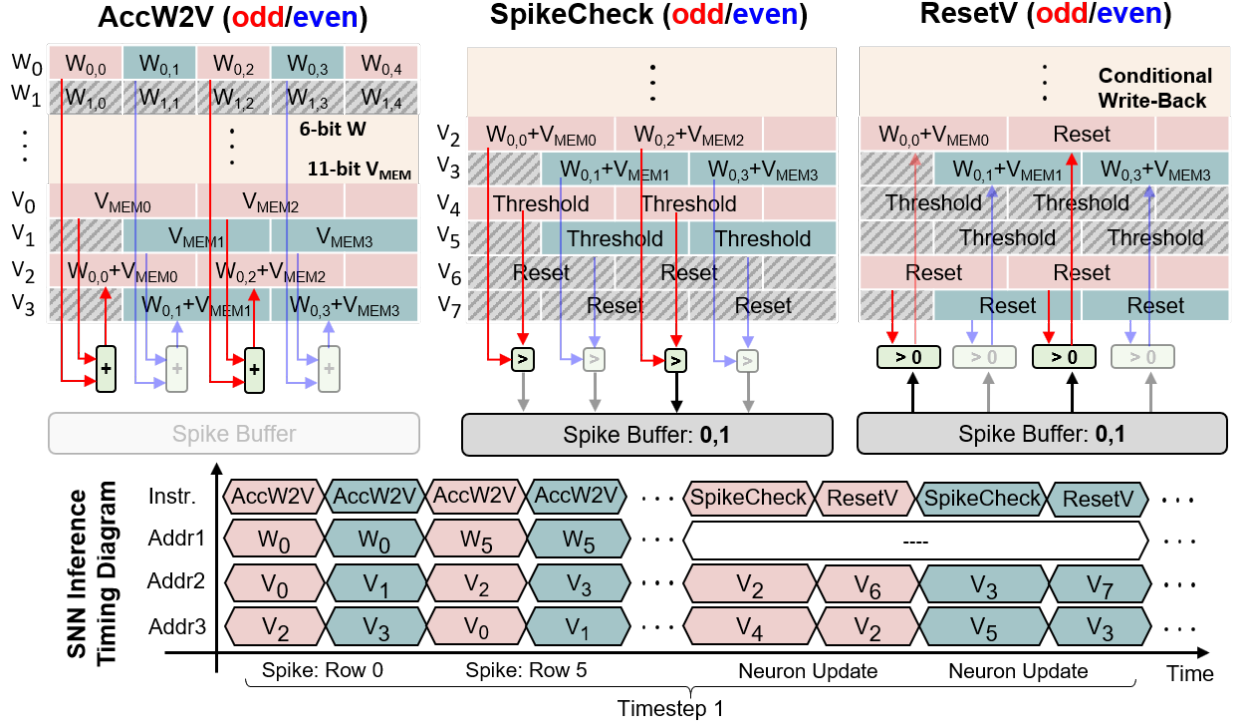
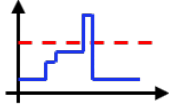
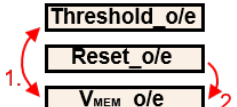
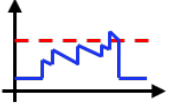
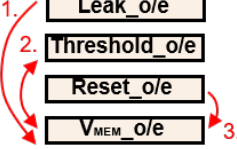
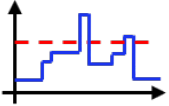
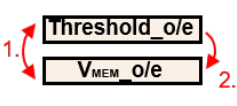


Figure 7.5. Illustration of supported in-memory SNN instructions.

during even cycle, Col[6-17] form one adder, Col[18-29] form another, and so on. It's worthwhile to note that the sixth bit of V_{MEM} aligns with MSB of the weight (W_{sign}), and needs to be kept '0' to correctly read W_{sign} (hence, 11-bit V_{MEM}). CS block forwards this W_{sign} to the next six column peripherals for performing the full 11-bit accumulate operation.

7.2.2 In-Memory SNN Instructions

Fig. 7.5 shows the supported in-memory SNN instructions. In Accumulate-W-to-V (AccW2V) instruction, depending on which input neuron spikes, the corresponding RWL from the W_{MEM} block (RWLo in odd cycle, RWLe in even cycle), and one RWL and one WWL from the V_{MEM} block are enabled simultaneously. Thus, 6-bit weights and 11-bit membrane potentials are accumulated and updated simultaneously, along the whole row. Similarly, AccV2V (not shown) accumulates two V_{MEM} rows. During the SpikeCheck instruction, two RWLs from the V_{MEM} blocks are enabled, one corresponding to the membrane potential to be checked, and other storing the threshold. The adders formed by the column

Neuron Type	Characteristics	Macro mapping	Instruction Sequence	Energy/update (pJ)*
Integrate-Fire (IF)			1. SpikeCheck 2. ResetV	1.81
Leaky-integrate-fire (LIF)			1. AccV2V (leak) 2. SpikeCheck 3. ResetV	2.67
Residual-membrane-potential (RMP)			1. SpikeCheck 2. AccV2V	1.68

* Measured at 200MHz @ 0.85V.

Figure 7.6. Multiple neurons can be implemented using in-memory instructions.

peripherals act as comparators in this case, by checking if the sum is greater or less than 0. This can be achieved by checking the COUT from MSB column peripheral, which is then utilized in setting the corresponding spike buffer if the membrane potential exceeds the threshold. Subsequently, the ResetV instruction follows the SpikeCheck instruction. During ResetV, one RWL and one WWL are enabled in the V_{MEM} block corresponding to the reset value and the destination membrane potential, respectively. The BLFA is bypassed in this instruction, and the reset value read in the SINV block gets directly transferred to the CWD block. The spike buffers determine whether the CWD drives the WBLs/WBLBs or leaves them precharged, thereby conditionally writing into only those membrane potentials in a row which have spiked. Thus, during the SNN inference, each input spike translates to AccW2V (odd and even) instruction for accumulating weights to membrane potentials. At the end of each timestep, the neuron output is computed using SpikeCheck and ResetV instructions, thereby generating output spikes. This process is repeated for all timesteps.

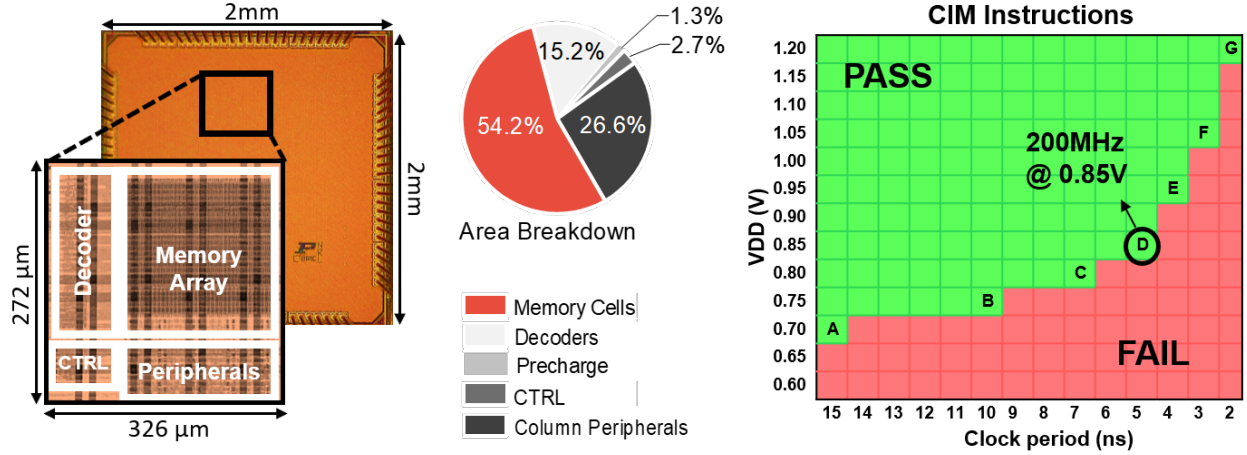


Figure 7.7. Die micrograph, area breakdown, and Shmoo plot for CIM operations.

7.2.3 Multiple Neuron Functionalities

Fig. 7.6 illustrates the characteristics of various neuron models, and how they can be implemented on IMPULSE through a combination of SpikeCheck, AccV2V, and ResetV instructions. The IF neuron can be implemented simply by using SpikeCheck and ResetV instructions as described in the previous section. The LIF neuron characteristics adds a ‘leak factor’, which can be incorporated by using AccV2V instruction to subtract the ‘leak’ value from the membrane potential, before using SpikeCheck and ResetV instructions. The RMP neuron [139], on the other hand, uses a soft reset, where the threshold value is subtracted from the membrane potential if it spikes, instead to resetting it. Thus, it can be implemented by using the AccV2V instruction after SpikeCheck, instead of ResetV. The figure also tabulates the measured energy per neuron-update at 200MHz clock and 0.85V supply.

7.3 Implementation Results

The prototype test chip was fabricated in 65nm CMOS technology (Fig. 7.7). It achieves a 54.2% memory area efficiency. Fig. 7.7 also shows the Shmoo plot for the CIM instructions. Fig. 7.8(a) plots the measured average power consumption and the energy-efficiency for AccW2V instruction (which is the main synaptic operation), for various possible operating points of interest identified on the CIM Shmoo (A-G). It can be observed that point D

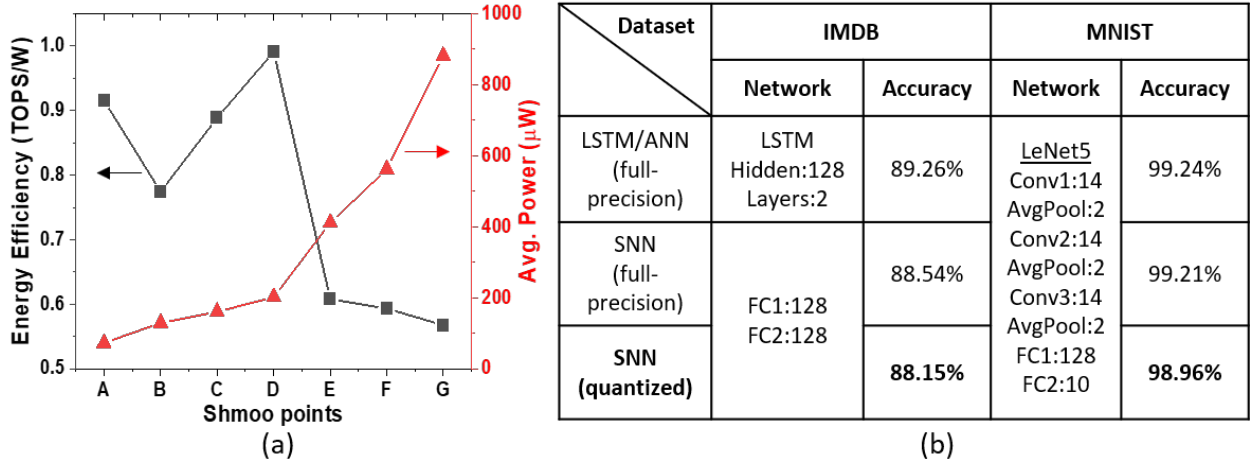


Figure 7.8. (a) Measured average power and energy-efficiency for AccW2V instruction. (b) SNN architecture and accuracy on IMDB and MNIST datasets.

(200MHz clock and 0.85V supply) achieves optimal energy-efficiency with 0.99 TOPS/W for AccW2V (1 op = 11-bit operation). Other instructions – AccV2V, ResetV, and SpikeCheck achieve 1.18, 1.02, and 1.22 TOPS/W, respectively, at point D.

We train an SNN with an input layer (100 neurons), two FC layers (128 neurons), and an output layer (1 neuron) to classify movie reviews from IMDB dataset [140]. SNN has 6-bit signed weights, and 11-bit V_{MEM} with RMP neurons. Each word in the sentence is converted to a 100-d vector, and presented to the SNN for 10 timesteps. SNN is trained with surrogate-gradient based backpropagation with threshold and leak optimization [135]. The input layer acts as spike-encoder and the two FC layers are mapped successively on IMPULSE. The macro runs one layer at a time: the output spikes of the n^{th} layer obtained from the chip are stored in the external memory and streamed back to compute the $n+1^{th}$ layer using FPGA. The SNN achieves an accuracy of 88.15%, close to a corresponding 2-layer LSTM network (Fig. 7.8(b)). Fig. 7.9 shows the dynamics of output layer neuron’s membrane potential capturing the sentiment as the words are presented. V_{MEM} value above zero represents positive sentiment and vice-versa. We also trained an SNN with modified LeNet5 architecture to demonstrate image classification from MNIST dataset and achieved 98.96% accuracy with 10 timesteps. The first Conv layer acts as a spike-encoder, while Conv2,3 and FC1,2 are successively mapped on IMPULSE. Note that input channels for

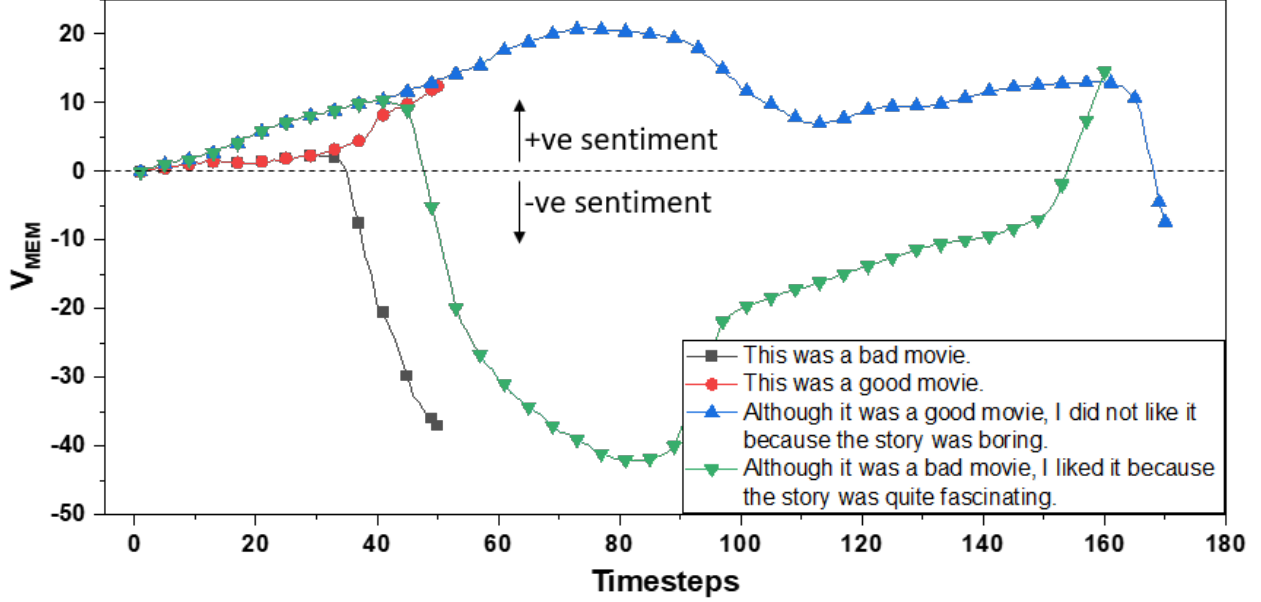


Figure 7.9. Progression of the final output neuron’s membrane potential with timesteps, where each word is presented to the SNN for 10 timesteps.

Conv layers were kept 14 with 3×3 kernel size to restrict the fan-in to 128 ($3 \times 3 \times 14 = 126$), to fit our macro. Similarly, the number of neurons in FC layers was kept ≤ 128 .

Fig. 7.10(a) plots the average sparsity in the spikes observed at each layer for each of the 10 timesteps. Note that these are averaged over each word and each image of the IMDB and MNIST test dataset, respectively. The overall sparsity of $\sim 85\%$ is achieved in both cases, which leads to significant energy improvements. The proposed macro exploits the input-spike sparsity in SNNs since the number of spikes determine the number and sequence of instructions executed. The measured EDP per-neuron per-timestep is plotted with varying sparsity (0%: all 128 input neurons spike, 100%: no input neuron spikes) showing a 97.4% reduction in EDP at 85% sparsity, as shown in Fig. 7.10(b).

We compare the energy-efficiency of our approach of CIM SNN with an LSTM implemented on an LSTM accelerator [136] in Table 7.1. The SNN has $8.5\times$ lower trainable parameters, and $5.6\times$ higher energy-efficiency per inference, when considering timesteps and the sparsity of input spikes. We also compare with other state-of-the-art SNN macros [141]–[143] and digital CIM macros [144]–[146] (Table 7.2). Among the SNN macros, [141] has a poor energy efficiency due to time-based digital oscillator circuits for implementing

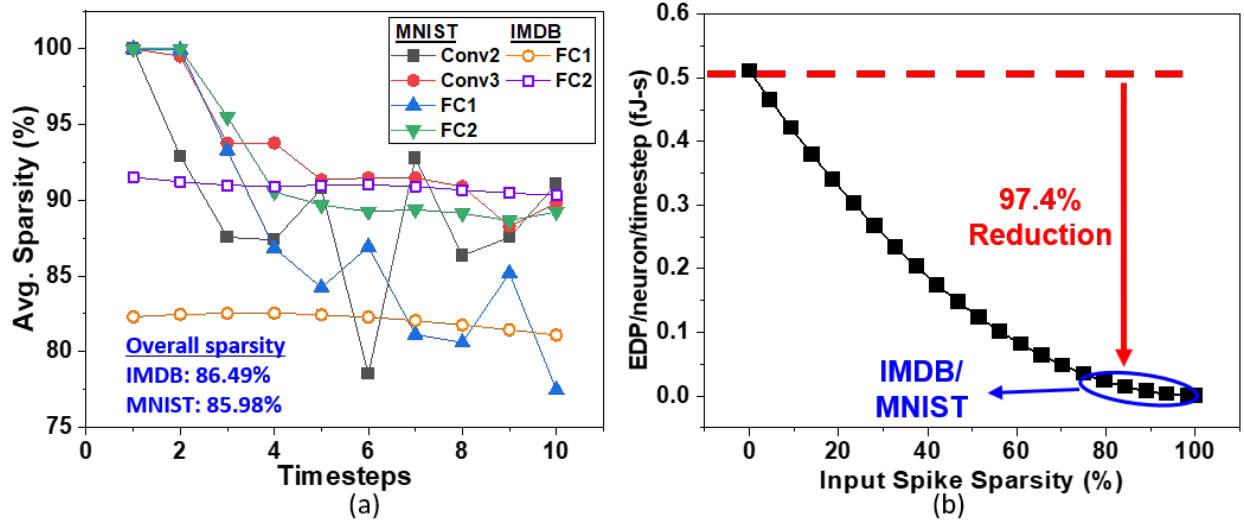


Figure 7.10. (a) Average sparsity obtained at each layer of SNN for each timestep. (b) Measured EDP per-neuron per-timestep with varying sparsity.

Table 7.1. Energy Efficiency of SNN over LSTM.

Network	Model	#Parameters	Timesteps	Sparsity	Effective #Ops	Energy/inf (nJ)
LSTM	2 hl, 128 n	247.8K	1	-	247.8K	246* [3]
SNN	100-128-128-1	29.3K	10	86.5%	39.5K	43.6

* Considers complete system energy

neuron functionality, while [143] has $2.7\times$ lower energy-efficiency (assuming linear scaling with bit-precision) compared to our design due to very low-frequency operation. Ref. [142] implemented an area-efficient weight memory, however, this design would still suffer from membrane potential bottlenecks. On the other hand, among digital CIM macros, 6T-SRAM based macro [144] achieves high memory area efficiency, however, it suffers from read disturb failures. Ref. [145] and [146] proposed 8T-SRAM based CIM macros developed for ANNs, having $1.5\times$ and $2.2\times$ lower energy-efficiency compared to our design, respectively. Thus, IMPULSE is the only digital CIM based SNN macro, with support for all instructions required for SNN inference, and multiple neuron functionalities. It also supports both FC and Conv layers and is scalable to larger networks by employing a distributed multi-macro architecture.

Table 7.2. Comparison with prior works.

	CICC'17 [7]	CICC'19 [8]	ASSCC'20 [9]	VLSI'15 [10]	ISSCC'19 [11]	VLSI'20 [12]	This Work
Technology	65 nm	28 nm	65 nm	28 nm	28 nm	65 nm	65 nm
Application	SNN	SNN	SNN	CAM/Logic	CNN/FC	CNN	SNN
Type	Time based	Digital	Async	CIM	CIM	CIM	CIM
Memory Capacity	8K synapses	64K synapses	67K synapses	4Kb	128Kb	38Kb	11.25Kb
Precision (W/V_{MEM})	3b/8b	4b/-	1b/6b	-	8b/-	16b/-	6b/11b
Bitcell Type	-	6T	-	6T	8T	8T	10T
Read Disturb	-	No	-	Yes	No	No	No
Flexible Neuron	No	No	No	No	No	No	Yes
Sparsity	No	No	Yes	No	No	Yes	Yes
Area (mm²)	0.24	0.266	1.99	0.0012	2.7	0.377	0.089
Supply (V)	1.2	1.1	0.5	1	0.6	1	0.85
Freq (MHz)	99	255	0.07	370	114	200	200
Power (mW)	20.48	1.023	0.0003	-	105	5.294	0.201
Performance (GOPS)	0.0062 ^{\$}	-	-	-	32.7	3.16	0.2
Performance/Area (GOPS/mm²)	1.65 [^]	-	-	-	27.3	8.4	2.24
Energy Efficiency (TOPS/W)	0.019 ["]	-	0.67 (6b)	-	0.97* (8b)	0.31 (16b) 4.91 (1b)	0.99 (11b)

* Scaled to 65nm assuming energy \propto (Tech.)² ^ GSpires/s/mm² \$GSpires/s " TSpires/s/W

7.4 Multi-macro Architecture

7.4.1 Introduction

A single macro alone has a limited fan-in. For example, considering the dimensions of IMPULSE shown in Fig. 7.11 (left), it has 128 weight rows which correspond to an input neuron fan-in of 128. Similarly it has 16 odd and even rows for V_{MEM} which correspond to 16 pixels in the output feature map. Each row has 72 columns of bits storing the weight values (6-bit each) which correspond to 12 different kernels or output channels in the output feature map. To process larger layers with higher fan-in, there is a need to think about how this macro can be scaled up.

We propose to have multiple such macros connected together, where the weights of a large layer are partitioned across these different macros, illustrated in Fig. 7.11 (right).

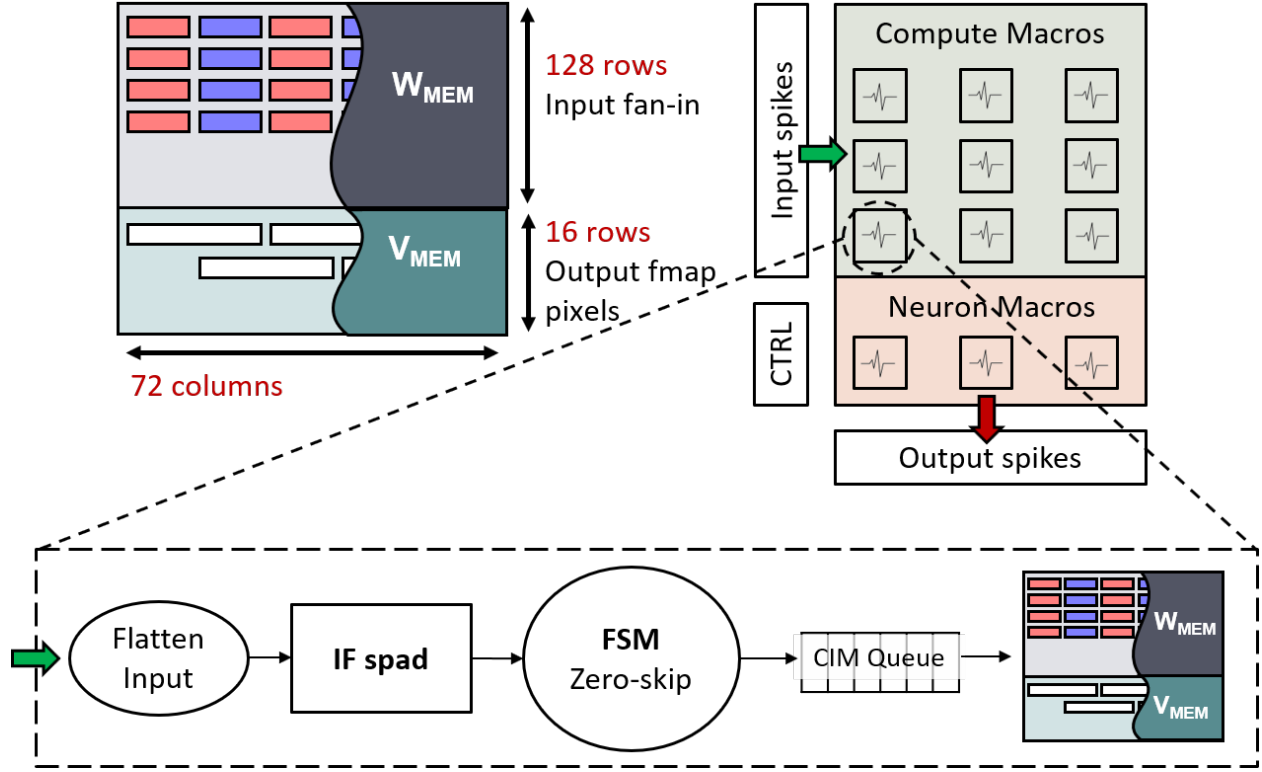


Figure 7.11. Limited fan-in of single macro (left). Multi-macro architecture is shown on the right consisting of compute and neuron macros and moving partial V_{MEM} among them to compute the final spikes (right). The block diagram showing each processing element (bottom).

Each macro performs a part of the computation, and then moves its partial V_{MEM} to the next macro to accumulate its partial V_{MEM} , and so on. Once the partial V_{MEM} have been moved among all macros, they are transferred to the neuron macros, which keeps track of the final V_{MEM} across timesteps and generates the output spikes.

Fig. 7.11 (bottom) shows the block diagram for each processing element. The input spike buffer stores the spikes for one layer at a time for all the timesteps. The first block reads the input spikes from this buffer and flattens them to arrange the spikes in the scratchpad (IF spad). There is a finite-state-machine (FSM) which implements the zero-skipping, reading the data from the IF spad and pushes the spiking events into the CIM queue, which are eventually processed in the macro.

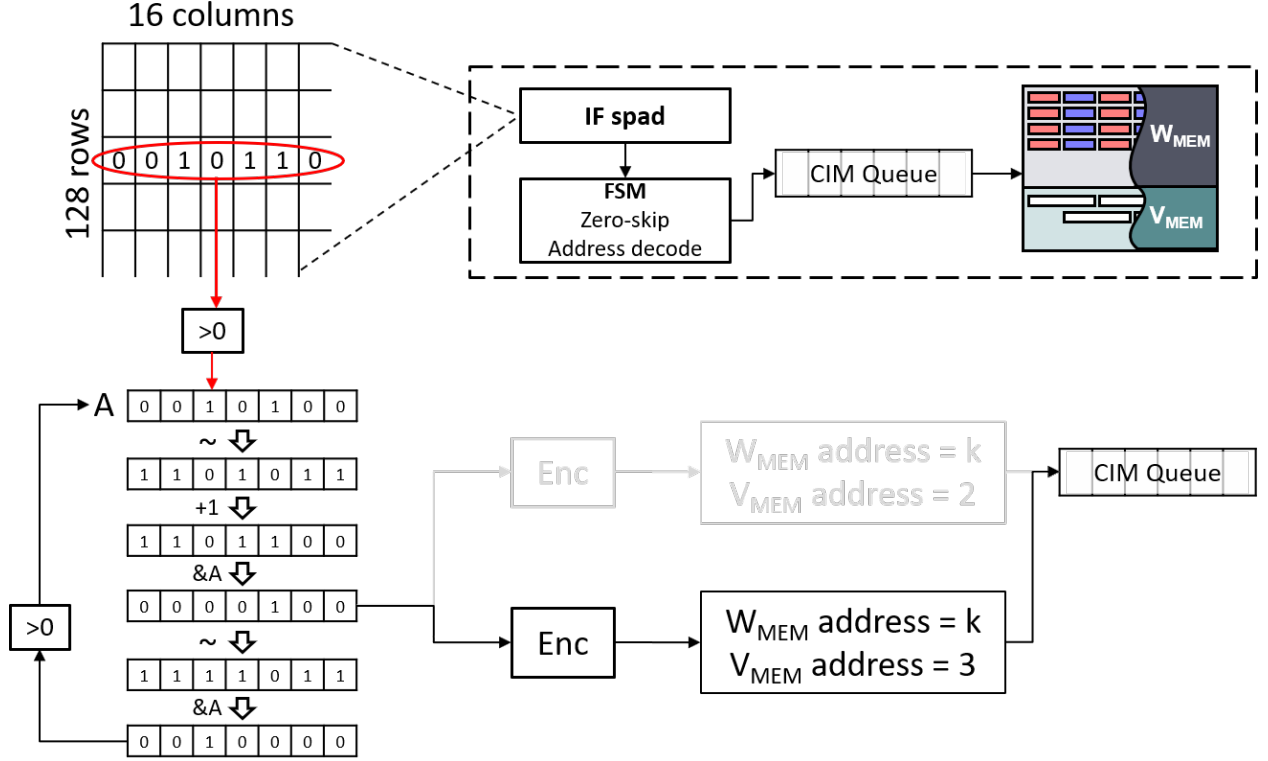


Figure 7.12. Zero-skipping using leading-one detector.

7.4.2 Zero-skipping

One of the key aspects of SNNs, which has a great potential for energy-efficiency, is the high-level of sparsity observed in the input spikes. This means, that our IF spad will have significant amount of zeros, whose compute steps can be skipped, leading to reduced overall energy and latency.

Looking at the IF spad (see Fig. 7.12), it has 128 rows and 16 columns. To implement the zero-skip, we need to extract out the row and column index for each entry which is ‘1’. The row index gives the W_{MEM} address while the column index gives the V_{MEM} address to be enabled for that spike in the macro. One straightforward way is to check all entries one by one, however, this is very slow and the macro will remain idle most of the time waiting for data. Thus, we propose to use the leading-one-detector (LOD) circuit to implement zero-skipping. One implementation of the LOD is illustrated in Fig. 7.12. We read entries from the IF spad row-by-row. The 16-bit vector is checked if all elements are zeros, and if not, it

is stored in a temporary register A. Following a series of bitwise operations such as NOT, ADD, AND etc., the leading-one can be separated out from the vector. Passing this through an encoder generates the required indexes which can be pushed into the CIM queue. To get the next leading one, we perform a couple more bitwise operations to remove the processed spike from the current vector. The new generated vector is checked for zero condition and fed back to the LOD circuit in the next cycle. This ensures that in every cycle data gets written into the CIM queue, so that the macro does not remain idle. Finally, this process repeats until all spikes are processed in the current row, and eventually in every row of the IF spad.

7.4.3 Macro Pipeline

As the CIM queue starts to fill up, the macro can start processing the events. We introduce pipeline stages (flip-flops) between the column peripheral blocks – SINV, BLFA, and CWD as shown in Fig. 7.13 (right). These blocks correspond to the read, compute and store pipeline stages, respectively. The corresponding timing diagram is shown on the left in the figure. The first entry from the queue is read and the corresponding W_{MEM} and V_{MEM} rows are enabled. Thus, in the first cycle, the data has been latched into the SINV (shown in yellow). In the next cycle, the data from SINV gets computed in the BLFA. While this happens, the memory array and the SINV block operates on the next data from the queue (shown in blue). Further, in the third cycle, the computed data moves to the CWD block, and the V_{MEM} address is enabled (write word line) to overwrite the updated data. While the second data moves from SINV to BLFA, and the memory array is operating on the third data (shown in green), and so on. The macro pipeline gets filled and continues to process the spikes as long as the queue is not empty. Note that once the pipeline is full, three addresses in the memory array are being activated simultaneously (two RWLs and one WWL). There can be corner cases (data-dependent) causing RWL and WWL conflict on the same row, thus, appropriate bubbles need to be inserted in the pipeline for correct functionality.

As discussed in previous sections, these operations need to be performed for both odd and even weight columns. However, re-configuring the macro for each odd and even cycle

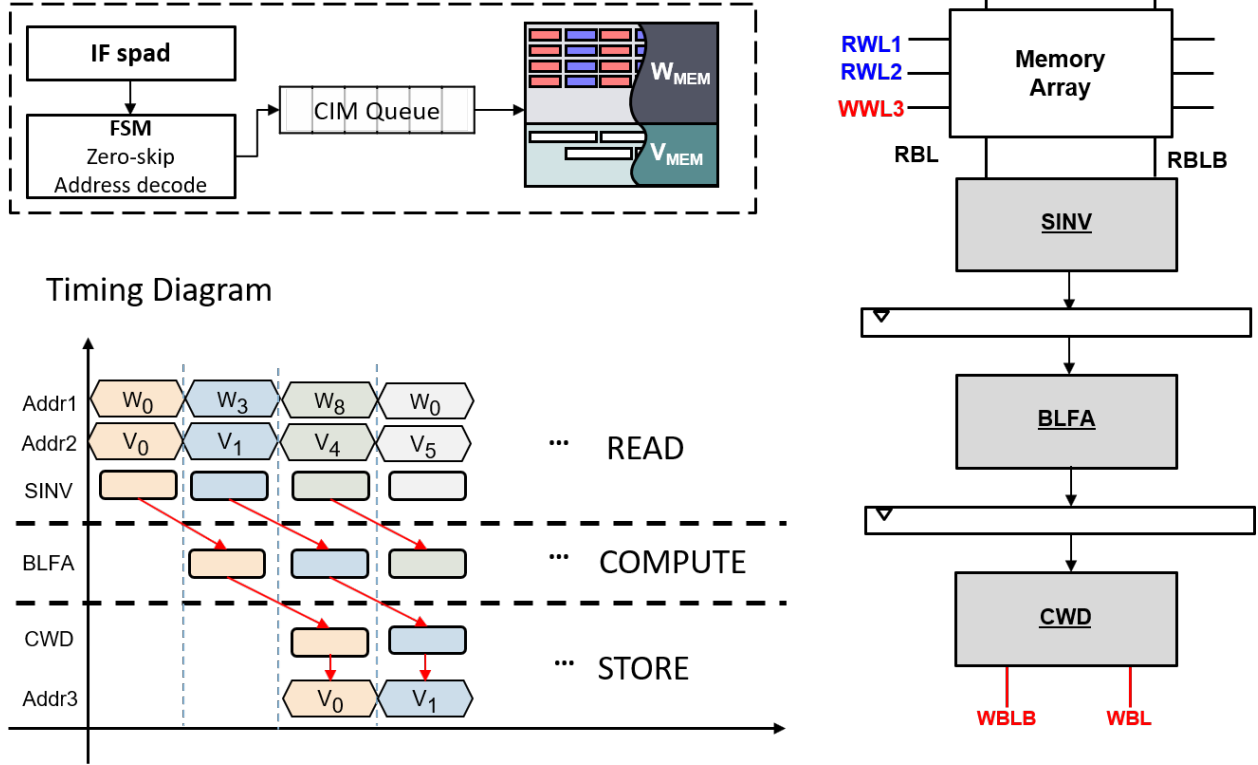


Figure 7.13. Macro Pipeline. Pipeline stages Read, Compute and Store corresponding to the column peripheral blocks SINV, BLFA and CWD, respectively (right). Timing diagram of the macro processing is shown on the left.

increases the switching activity, which hurts the overall energy efficiency. Fig. 7.14 (right) plots the normalized energy per operation, as a function of how many consecutive odd or even cycles we do. For example, if we interleave odd and even for each cycle (as shown in Fig. 7.14 (top left)), this would correspond to ‘1’ on the x-axis. Since this scenario has the maximum switching activity, its energy is also high. We can observe from the plot that as we increase the number of consecutive even/odd cycles, the energy per operation reduces. This is because the overhead due to the switching activity is now amortized over multiple consecutive operations. This can be implemented on our processing element as shown in Fig. 7.14 (bottom left). We have two CIM queues, one for even and one for odd, and having a MUX select from which queue to perform the operation in the macro. The depth of these queues will determine the number of consecutive odd/even operations being performed, and can be chosen by looking at the energy plot.

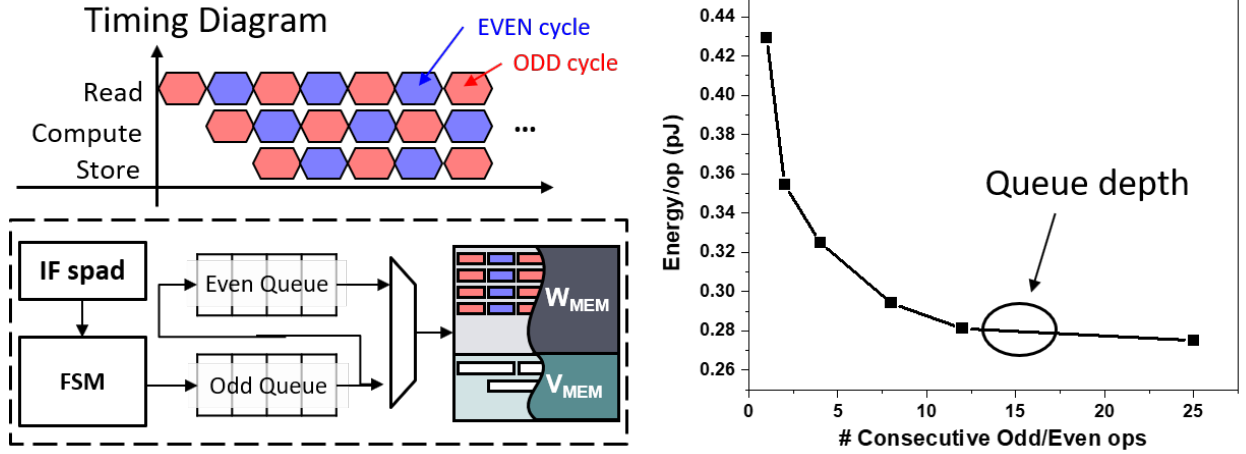


Figure 7.14. Amortizing the reconfigurability overhead for processing odd and even columns, by maintaining two queues having appropriate depth. The graph on the right plots the energy per operation as a function of number of consecutive odd/even operations.

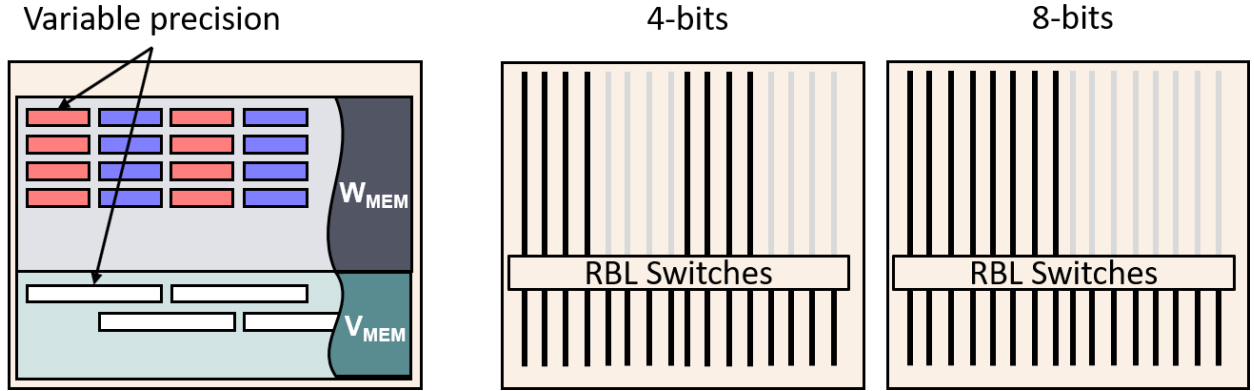


Figure 7.15. Variable bit-precisions can be supported by introducing RBL switches to decouple the RBL between the W_{MEM} and V_{MEM} subarrays.

7.4.4 Support for Multiple Bit-precision

The macro described above had fixed weight precision of 6-bit and V_{MEM} precision of 11-bit. This was hard-coded in design by having dual odd/even RWLs in each row. In order to extend the capability of the macro to support multiple precision, we propose to add read-bitline switches to decouple the RBLs between W_{MEM} and V_{MEM} subarrays of the memory, as illustrated in Fig. 7.15. The RBL switches can be re-configured based on the precision.

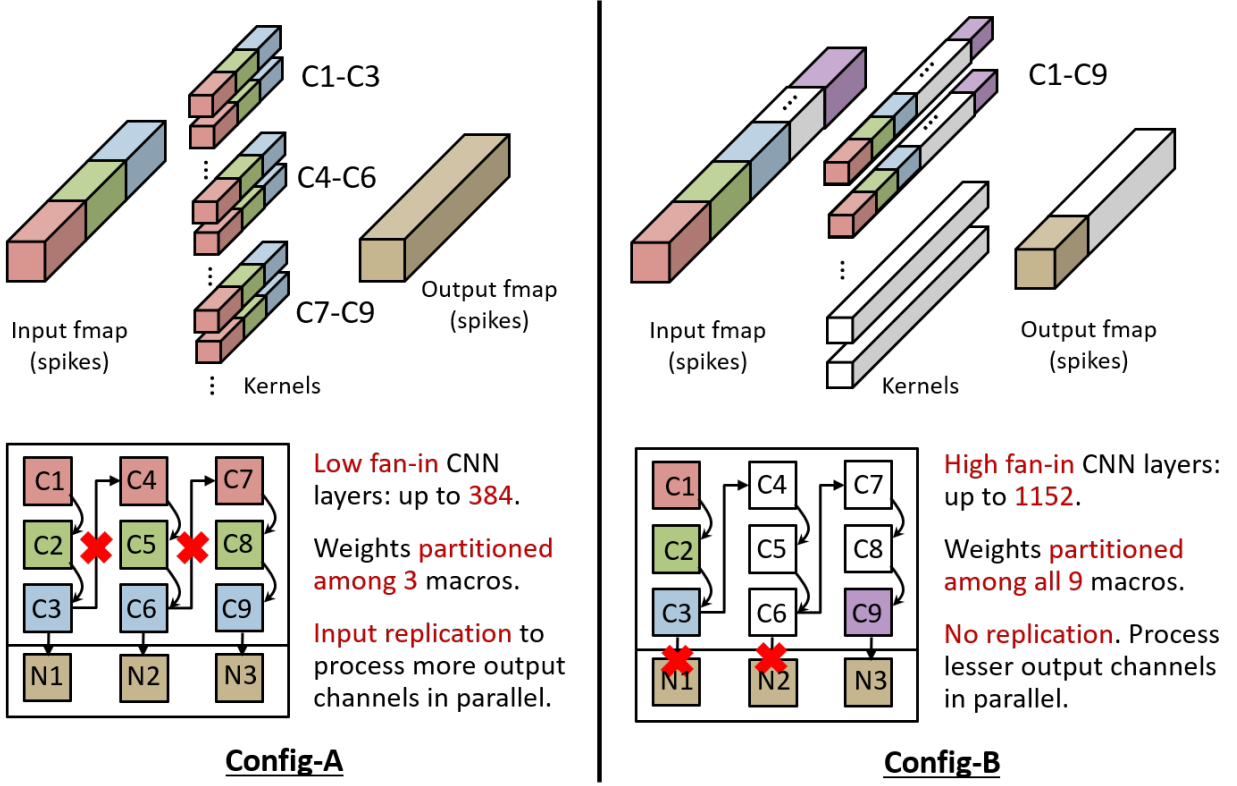


Figure 7.16. Two supported configurations to efficiently run low fan-in and high fan-in CNN layers with high throughput.

For example, to have 4-bit weights and 7-bit V_{MEM} , the RBL switches connect the RBLs in the first four columns, and disconnect the RBLs in next four and so on, during the odd cycle, and vice-versa during the even cycle. Similarly, for running 8-bit weights and 15-bit V_{MEM} , first eight columns are connected during odd, and next eight during even, and so on. The column peripherals are also reconfigured based on the precision using the CMUXes described earlier. Thus, now with 72 columns in our memory array, we can store and process 18, 12 and 9 output channels at once for 4-bit, 6-bit and 8-bit weight precision, respectively. This also eliminates the need for having dual RWLs and relaxes the memory layout design.

7.4.5 Configurations for Low and High Fan-in CNN Layers

To efficiently implement both low fan-in and high fan-in CNN layers, we support two configurations of data-movement in our proposed architecture, which are summarized in

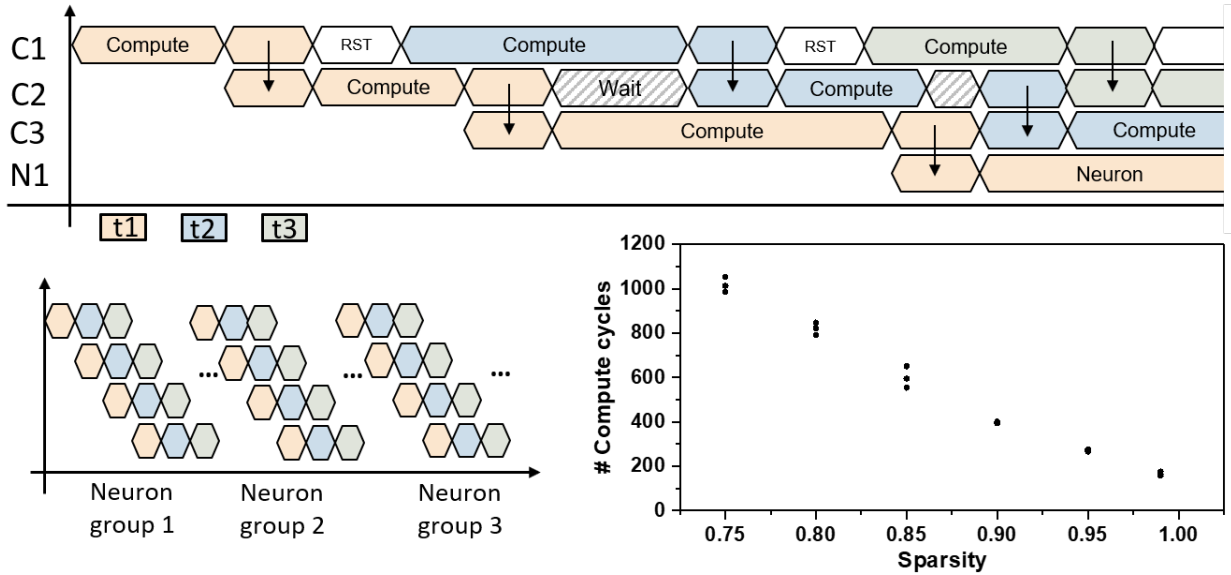


Figure 7.17. Timing diagram showing the top-level timestep pipelining among the macros (top). Once all timesteps are performed for one group of neurons, we can start processing the next group and so on, thereby leveraging weight re-use across neurons and across timesteps. The plot (bottom right) shows the dependency of compute cycles on the data-sparsity, showing the benefits of zero-skipping.

Fig. 7.16. In Config-A, low fan-in CNN layers (upto 384 input fan-in) can be run. The weights and inputs are partitioned among 3 macros (C1-C3) and the partial V_{MEM} are transferred from $C1 \rightarrow C2 \rightarrow C3 \rightarrow N1$. To increase the throughput, we do input replication to process more output channels in parallel. Thus, C4-C6 and C7-C9 receive the same inputs as C1-C3, but they store different kernels. On the other hand, Config-B can support fan-in upto 1152. In this case, the weights are partitioned among all nine compute macros, and the partial V_{MEM} moves from C1 all the way to C9, before getting computed in N3. Thus, in this configuration, we can process higher fan-in layers, but lesser number of output channels at once, giving us equivalent throughput as Config-A. Note that these configurations extend to low and high fan-in FC layers as well.

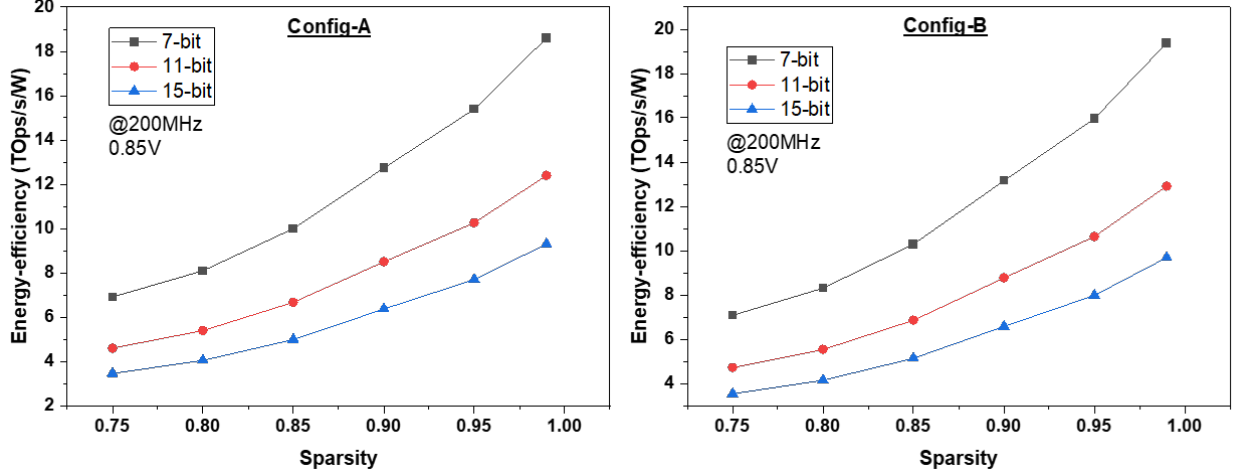


Figure 7.18. Energy-efficiency of the proposed multi-macro architecture, as a function of data-sparsity and the bit-precision, for the two proposed configurations.

7.4.6 Timestep Pipelining: Leveraging Additional Weight Re-use

At the top-level, we pipeline the macros across timesteps to leverage the additional weight re-use in SNNs. Fig. 7.17 (top) illustrates the timing diagram, taking the example for Config-A. We start with C1 computing timestep one (t_1) shown in orange. Once all the spikes are computed, the partial V_{MEM} is transferred from C1 to C2, and C2 starts its compute for t_1 . Meanwhile, C1 becomes free to start computing t_2 . As the pipeline fills, different macros compute different timesteps for the same neuron group. Once all the timesteps have been computed for one neuron group, the process can repeat for the next group of neurons (see Fig. 7.17 (bottom left)). Note that the weights stored in the IMPULSE memory are not changed and are held stationary, thereby leveraging weight re-use across neurons and across timesteps. Fig. 7.17 (bottom right) shows the dependency of compute cycles on the input spike sparsity, showing the benefits of zero-skipping. Since the compute cycles are highly data-dependent, we use asynchronous handshaking protocol using a global controller to manage the instructions running on each macro in parallel.

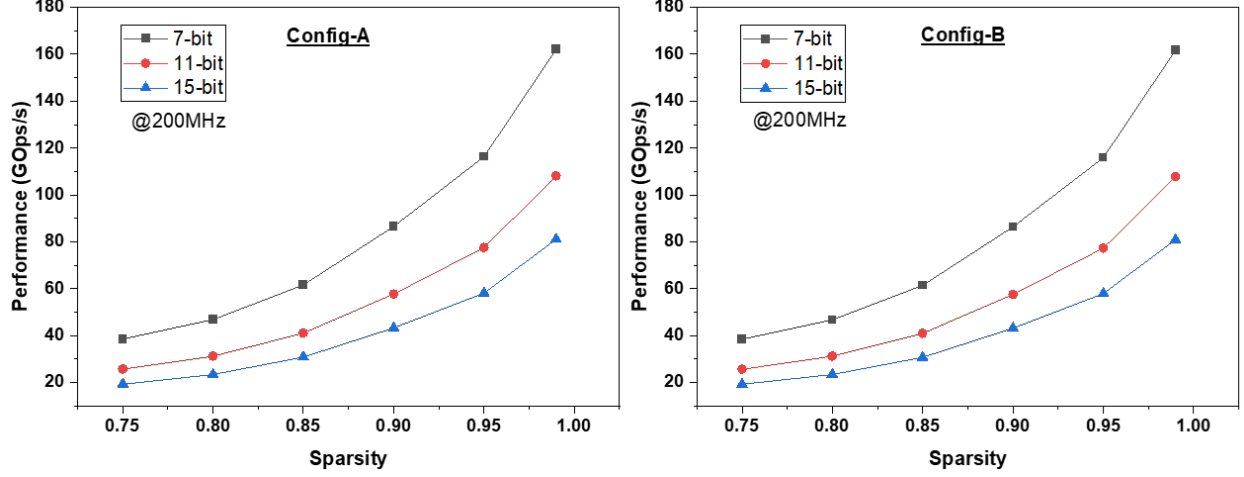


Figure 7.19. Throughput of the proposed multi-macro architecture, as a function of data-sparsity and the bit-precision, for the two proposed configurations.

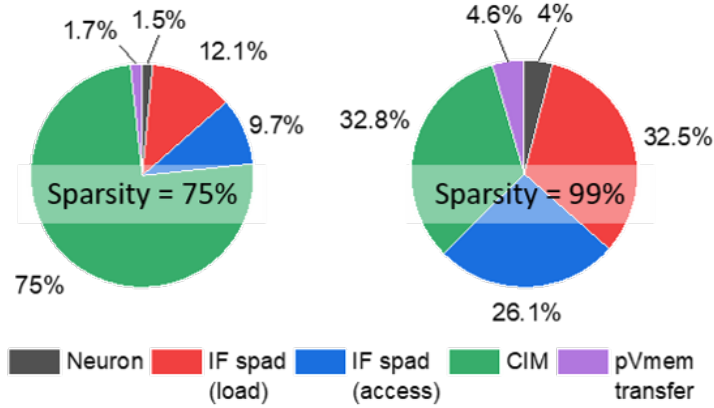


Figure 7.20. Energy breakdown for two levels of sparsity (75% and 99%).

7.4.7 Preliminary Results

Fig. 7.18 and Fig. 7.19 plots the simulation results using the 65nm process technology for the energy-efficiency (TOPs/s/W) and throughput (GOps/s), respectively, of the proposed multi-macro architecture. For Config-A, a CNN layer of 3x3 kernels with input feature map of 16x16x42 was chosen, which has a fan-in of 378 ($=3 \times 3 \times 42$). For Config-B, a CNN layer of 3x3 kernels with input feature map of 16x16x128 was chosen, which has a fan-in of 1152 ($=3 \times 3 \times 128$). The following observations can be made from the results:

1. Both configurations have similar metrics when normalized to energy and performance per operation, since both configurations have the same effective number of operations (low fan-in high fan-out in Config-A and vice-versa in Config-B).
2. We achieve a better throughput and energy-efficiency as the sparsity in input spikes increases. This is a direct consequence of zero-skipping, which leads to lesser number of operations being run on the IMPULSE macro. Hence, it also lowers the overall compute cycles.
3. As the weight/ V_{MEM} bit-precision is reduced, the peripherals are reconfigured to more number of adders working in parallel, which leads to more parallelism, thereby better energy-efficiency and throughput.

Fig. 7.20 shows the energy breakdown. For a low sparsity case, it can be observed that majority of the energy is dominated by the CIM operations, while about a fourth of the energy is consumed in accessing the input spikes, neuron operations, and partial V_{MEM} movements. Even at very high sparsity ($\sim 99\%$), at least a third of the total energy is still dominated by the CIM operations. Since the number of CIM operations are less at such a high sparsity, the component of data-movements compared to the CIM operations is higher in the energy distribution.

7.5 Conclusion

We present a digital CIM macro with fused weights and membrane potential, designed for efficient processing of SNN inference. The proposed macro inherently leverages the sparsity and supports multiple neuron functionalities. An optimal energy-efficiency of 0.99 TOPS/W was achieved for 11-bit signed operations. We demonstrate our approach by training an SNN for a sentiment analysis task utilizing the intrinsic dynamics of V_{MEM} , and also for an image classification task, achieving competitive accuracy to their corresponding LSTM and ANN counterpart, respectively.

Further, we extend the macro for processing larger layers and propose a multi-macro architecture. It supports two configurations to efficiently process both low fan-in and high fan-

in layers with high throughput. Moreover, we leverage the additional weight re-use in SNNs by incorporating timestep pipelining. Each processing element incorporates zero-skipping to leverage the high-sparsity of input spikes in SNN workloads for reducing the energy and latency. The macro can be reconfigured to support various bit-precisions for weights/ V_{MEM} . We also proposed micro-architecture extensions to amortize the reconfigurability overheads on IMPULSE macro.

8. SUMMARY AND FUTURE DIRECTIONS

In memory computing is an emerging paradigm which tries to address the memory access bottleneck which is becoming prominent in recent workloads such as machine learning and artificial intelligence. By enabling computations within the memory arrays where the data is stored, one can perform massively-parallel operations by exploiting the high internal memory bandwidth. Moreover, since less data transfers occur from memory to the processing units, this approach leads to energy-efficiency.

In this dissertation, we proposed various circuit techniques to enable computations in standard SRAMs. Given the read-stability issue arising from 6T-SRAM cells, we looked at two-port register file such as 8T, 9T and 10T SRAM bitcell configurations in our toolbox. We proposed how bitwise Boolean operations, binary convolution operations, dot-product operations, lookup table based computing, spiking neural networks, etc., can be performed in the SRAM arrays. These circuit explorations show that on-chip SRAMs can do much more than just store data and can be re-purposed as on-demand accelerators for a variety of applications.

Looking ahead, there is really a need to understand the CIM implications across the stack – devices, circuits, systems and algorithms. With promising results obtained from the circuit primitives which were extensively explored in this dissertation, it is worth exploring the systems aspects where optimal mapping, scheduling, data-flow etc., can be considered for implementing various workloads on CIM primitives. In addition, a hardware-software co-design needs to be developed, where the algorithm can be modified such that when running on CIM hardware, it achieves peak efficiency and compensates for errors introduced due to approximate computing on CIM primitives. On the other hand, there is extensive device research especially on “computational devices”, where the device characteristics itself mimics certain functionalities, or store data in its conductive states, which can be used for computing in the analog domain. With many such devices being developed, especially the embedded non-volatile memories such as RRAM, PCM, FeFET, MRAM etc., there is a need to explore the heterogeneous integration of such devices in the CMOS process. Most circuit techniques described in this dissertation also apply to these emerging memory technologies.

A. CHALLENGES WITH 6T SRAM FOR ENABLING COMPUTE-IN-MEMORY

A.1 Operation of 6T SRAM

The most popular and widely used SRAM consists of six transistor bitcell. Fig. A.1 shows the schematic of the 6T-SRAM bitcell. It consists of two cross-coupled inverters and two access transistors AXL and AXR, which connect the inverters to the bitlines BL and BLB, respectively. During the write operation, BL and BLB are set to V_{DD} and GND, respectively, for writing ‘1’ to the cell, and vice-versa for writing a ‘0’. Once the bitlines are set to the corresponding voltages, the wordline is enabled, which turns the access transistors ON and the data gets written into the cross-coupled inverter pair. All cells in the same row can be written at once, by setting the corresponding bitlines. For the read operation, both bitlines BL and BLB are precharged to V_{DD} and left floating, followed by enabling the wordline for the corresponding row that needs to be read out. The BL or BLB discharges through AXL or AXR, if the data stored in the cell is ‘0’ or ‘1’, respectively. Thus, a voltage difference is achieved between the BL and BLB which can be sensed using the sensing amplifiers connected to the peripheral circuitry.

A.2 Read Stability Challenges due to CIM

For enabling in-memory computing, multiple wordlines are enabled during the read operation, such that the data from multiple bitcells can interact on the bitlines to give us a logical operation. However, doing this raises some read stability issues.

A.2.1 Short-circuit paths

Consider a case shown in Fig. A.1, where the cell A0 stores ‘1’ and cell B0 stores ‘0’. Enabling both WL1 and WL2 to sense the logical output of A0 and B0 on the bitlines in this case would cause a short circuit path shown by the red dotted arrows. This causes a high current flowing through the storage nodes which can lead to unknown voltages depending

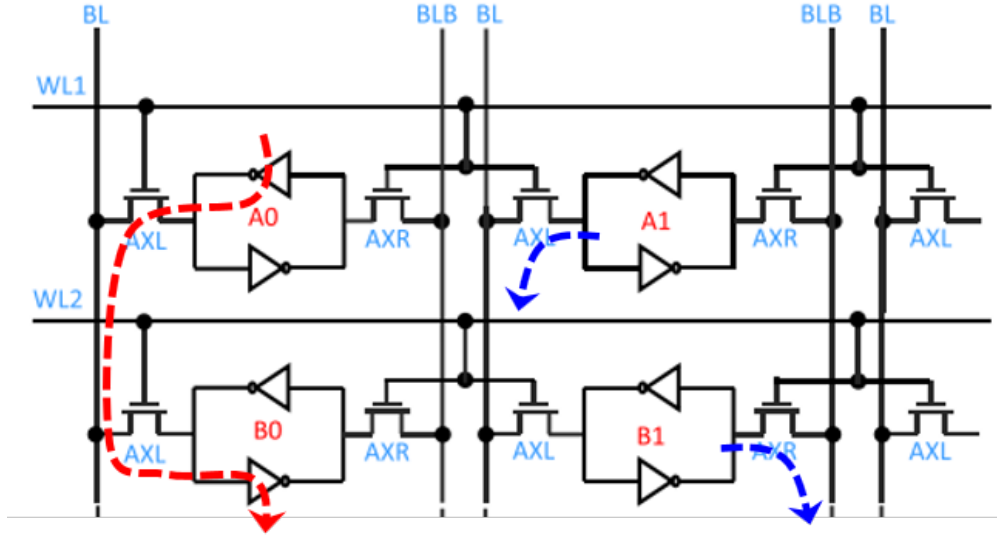


Figure A.1. Schematic showing a 6T-SRAM array. The red and blue arrows show the current path and the charge discharge path, respectively, causing read-stability issues for performing in-memory computing.

on the process variations. Thus, it not only has reliability concerns, but reduces the read stability of the cell, which might cause unintended data flips.

There are certain solutions which have been tried to avoid this problem. First, wordlines can be driven at a lower voltage to reduce the magnitude of high currents [11]. However, this requires generating multiple voltages, and modifications to the peripheral circuitry to work with multiple supply voltages. Moreover, this would also increase the latency of the read operation.

Another approach is to stagger the wordline activations [90]. For example, as shown in Fig. A.2, the wordlines for different rows are activated one after the other. Although this technique avoids the high current paths, it also increases the read latency, and suffers from timing challenges. The discharge on the bitlines is leakage dependent which can vary quite a bit with process variations.

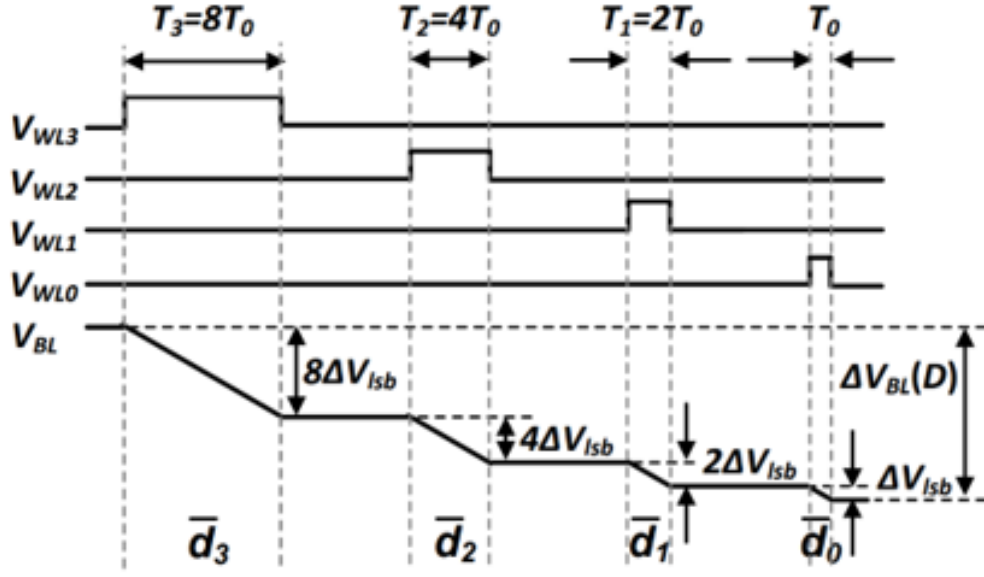


Figure A.2. Staggered activation of wordlines to avoid short circuit paths, leading to bitline discharge [90]

A.2.2 Pseudo-write

Another challenge which arises is the *pseudo-write*. For a typical read operation, only one of the bitlines discharge, while the other always is help at V_{DD} . This helps with read stability, and the 6T-SRAM bitcell is sized appropriately to handle one of the bitlines getting discharged. Now, consider the case where A0 stores ‘1’ and cell B0 stores ‘0’ (see Fig. A.1). If we activate the wordlines (either together or staggered), both BL and BLB will discharge, as shown by blue dotted arrows. This appears like a write-operation and in case the discharge becomes more (due to process variations), it would lead to data flips.

REFERENCES

- [1] Y. Bengio *et al.*, “Learning deep architectures for AI,” *Foundations and trends® in Machine Learning*, vol. 2, no. 1, pp. 1–127, 2009.
- [2] N. Jones, “The learning machines,” *Nature*, vol. 505, no. 7482, p. 146, 2014.
- [3] V. Sze, Y.-H. Chen, T.-J. Yang, and J. S. Emer, “Efficient processing of deep neural networks: A tutorial and survey,” *Proceedings of the IEEE*, vol. 105, no. 12, pp. 2295–2329, 2017.
- [4] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
- [5] J. Backus, “Can programming be liberated from the von neumann style?: A functional style and its algebra of programs,” *Commun. ACM*, vol. 21, no. 8, pp. 613–641, Aug. 1978, ISSN: 0001-0782. DOI: [10.1145/359576.359579](https://doi.org/10.1145/359576.359579).
- [6] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.
- [7] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.
- [8] F. Li and B. Liu, “Ternary weight networks,” *ArXiv*, vol. abs/1605.04711, 2016.
- [9] C. Zhu, S. Han, H. Mao, and W. J. Dally, “Trained ternary quantization,” *arXiv preprint arXiv:1612.01064*, 2016.
- [10] N. Verma, H. Jia, H. Valavi, Y. Tang, M. Ozatay, L. Chen, B. Zhang, and P. Deaville, “In-memory computing: Advances and prospects,” *IEEE Solid-State Circuits Magazine*, vol. 11, no. 3, pp. 43–55, 2019.
- [11] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, “A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory,” *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, Apr. 2016, ISSN: 0018-9200. DOI: [10.1109/JSSC.2016.2515510](https://doi.org/10.1109/JSSC.2016.2515510).
- [12] S. Aga, S. Jeloka, A. Subramaniyan, S. Narayanasamy, D. Blaauw, and R. Das, “Compute caches,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2017. DOI: [10.1109/hpca.2017.21](https://doi.org/10.1109/hpca.2017.21).

- [13] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, “An energy-efficient VLSI architecture for pattern recognition via deep embedding of computation in SRAM,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, May 2014. DOI: [10.1109/icassp.2014.6855225](https://doi.org/10.1109/icassp.2014.6855225).
- [14] M. Kang, E. P. Kim, M.-s. Keel, and N. R. Shanbhag, “Energy-efficient and high throughput sparse distributed memory architecture,” in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, IEEE, May 2015. DOI: [10.1109/iscas.2015.7169194](https://doi.org/10.1109/iscas.2015.7169194).
- [15] W. M. Snelgrove, M. Stumm, D. Elliott, R. McKenzie, and C. Cojocaru, “Computational ram: Implementing processors in memory,” *IEEE Design & Test of Computers*, vol. 16, pp. 32–41, 1999.
- [16] J. Zhang, Z. Wang, and N. Verma, “In-memory computation of a machine-learning classifier in a standard 6t SRAM array,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 4, pp. 915–924, Apr. 2017. DOI: [10.1109/jssc.2016.2642198](https://doi.org/10.1109/jssc.2016.2642198).
- [17] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester, “A 0.3v VDDmin 4+2t SRAM for searching and in-memory computing using 55nm DDC technology,” in *2017 Symposium on VLSI Circuits*, IEEE, Jun. 2017. DOI: [10.23919/vlsic.2017.8008465](https://doi.org/10.23919/vlsic.2017.8008465).
- [18] M. Huang, M. Mehalel, R. Arvapalli, and S. He, “An energy efficient 32-nm 20-mb shared on-die l3 cache for intel® xeon® processor e5 family,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1954–1962, 2013.
- [19] J. Burgess, “Rtx on – the nvidia turing gpu,” in *2019 IEEE Hot Chips 31 Symposium (HCS)*, 2019, pp. 1–27. DOI: [10.1109/HOTCHIPS.2019.8875651](https://doi.org/10.1109/HOTCHIPS.2019.8875651).
- [20] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, *et al.*, “In-datacenter performance analysis of a tensor processing unit,” in *Proceedings of the 44th annual international symposium on computer architecture*, 2017, pp. 1–12.
- [21] Y. Chen, T. Krishna, J. S. Emer, and V. Sze, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.

- [22] N. P. Jouppi, C. Young, N. Patil, D. Patterson, G. Agrawal, R. Bajwa, S. Bates, S. Bhatia, N. Boden, A. Borchers, R. Boyle, P.-l. Cantin, C. Chao, C. Clark, J. Coriell, M. Daley, M. Dau, J. Dean, B. Gelb, T. V. Ghaemmaghami, R. Gottipati, W. Gulland, R. Hagmann, C. R. Ho, D. Hogberg, J. Hu, R. Hundt, D. Hurt, J. Ibarz, A. Jaffey, A. Jaworski, A. Kaplan, H. Khaitan, D. Killebrew, A. Koch, N. Kumar, S. Lacy, J. Laudon, J. Law, D. Le, C. Leary, Z. Liu, K. Lucke, A. Lundin, G. MacKean, A. Maggiore, M. Mahony, K. Miller, R. Nagarajan, R. Narayanaswami, R. Ni, K. Nix, T. Norrie, M. Omernick, N. Penukonda, A. Phelps, J. Ross, M. Ross, A. Salek, E. Samadiani, C. Severn, G. Sizikov, M. Snellham, J. Souter, D. Steinberg, A. Swing, M. Tan, G. Thorson, B. Tian, H. Toma, E. Tuttle, V. Vasudevan, R. Walter, W. Wang, E. Wilcox, and D. H. Yoon, “In-datacenter performance analysis of a tensor processing unit,” *SIGARCH Comput. Archit. News*, vol. 45, no. 2, pp. 1–12, Jun. 2017, ISSN: 0163-5964. DOI: [10.1145/3140659.3080246](https://doi.org/10.1145/3140659.3080246). [Online]. Available: <https://doi.org/10.1145/3140659.3080246>.
- [23] Chen, Yu-Hsin and Krishna, Tushar and Emer, Joel and Sze, Vivienne, “Eyeriss: An Energy-Efficient Reconfigurable Accelerator for Deep Convolutional Neural Networks,” in *IEEE International Solid-State Circuits Conference, ISSCC 2016, Digest of Technical Papers*, 2016, 262–263.
- [24] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, “X-sram: Enabling in-memory boolean computations in cmos static random access memories,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 65, no. 12, pp. 4219–4232, 2018.
- [25] H.-S. P. Wong and S. Salahuddin, “Memory leads the way to better computing,” *Nature Nanotechnology*, vol. 10, no. 3, pp. 191–194, Mar. 2015. DOI: [10.1038/nnano.2015.29](https://doi.org/10.1038/nnano.2015.29).
- [26] S. Shirinzadeh, M. Soeken, P.-E. Gaillardon, and R. Drechsler, “Fast logic synthesis for RRAM-based in-memory computing using majority-inverter graphs,” in *Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Research Publishing Services, 2016. DOI: [10.3850/9783981537079_0771](https://doi.org/10.3850/9783981537079_0771).
- [27] A. Agrawal, C. Lee, and K. Roy, “X-changr: Changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep neural networks,” *arXiv preprint arXiv:1907.00285*, 2019.
- [28] I. Chakraborty, M. Ali, A. Ankit, S. Jain, S. Roy, S. Sridharan, A. Agrawal, A. Raghunathan, and K. Roy, “Resistive crossbars as approximate hardware building blocks for machine learning: Opportunities and challenges,” *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2276–2310, 2020.

- [29] S. Jain and A. Raghunathan, “Cxdnn: Hardware-software compensation methods for deep neural networks on resistive crossbar systems,” *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 18, no. 6, pp. 1–23, 2019.
- [30] 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.
- [31] A. Ankit, I. E. Hajj, S. R. Chalamalasetti, G. Ndu, M. Foltin, R. S. Williams, P. Faraboschi, J. P. Strachan, K. Roy, and D. S. Milojicic, “Puma: A programmable ultra-efficient memristor-based accelerator for machine learning inference,” *arXiv preprint arXiv:1901.10351*, 2019.
- [32] 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,” in *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, IEEE, Jun. 2016. DOI: [10.1109/isca.2016.12](https://doi.org/10.1109/isca.2016.12).
- [33] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, “Computing in memory with spin-transfer torque magnetic ram,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, Mar. 2018, ISSN: 1063-8210. DOI: [10.1109/TVLSI.2017.2776954](https://doi.org/10.1109/TVLSI.2017.2776954).
- [34] W. Kang, H. Wang, Z. Wang, Y. Zhang, and W. Zhao, “In-memory processing paradigm for bitwise logic operations in stt-mram,” *IEEE Transactions on Magnetics*, 2017.
- [35] D. Lee, X. Fong, and K. Roy, “R-MRAM: A ROM-embedded STT MRAM cache,” *IEEE Electron Device Letters*, vol. 34, no. 10, pp. 1256–1258, Oct. 2013. DOI: [10.1109/led.2013.2279137](https://doi.org/10.1109/led.2013.2279137).
- [36] A. Jaiswal, A. Agrawal, and K. Roy, “In-situ, in-memory stateful vector logic operations based on voltage controlled magnetic anisotropy,” *Scientific reports*, vol. 8, no. 1, pp. 1–12, 2018.
- [37] I. Chakraborty, A. Agrawal, A. Jaiswal, G. Srinivasan, and K. Roy, “In situ unsupervised learning using stochastic switching in magneto-electric magnetic tunnel junctions,” *Philosophical Transactions of the Royal Society A*, vol. 378, no. 2164, p. 20190157, 2020.

- [38] A. Agrawal, I. Chakraborty, D. Roy, U. Saxena, S. Sharmin, M. Koo, Y. Shim, G. Srinivasan, C. Liyanagedera, A. Sengupta, *et al.*, “Revisiting stochastic computing in the era of nanoscale nonvolatile technologies,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2481–2494, 2020.
- [39] A. Agrawal, C. Wang, T. Sharma, and K. Roy, “Magnetoresistive circuits and systems: Embedded non-volatile memory to crossbar arrays,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, 2021.
- [40] A. Sebastian, T. Tuma, N. Papandreou, M. L. Gallo, L. Kull, T. Parnell, and E. Eleftheriou, “Temporal correlation detection using computational phase-change memory,” *Nature Communications*, vol. 8, no. 1, Oct. 2017. DOI: [10.1038/s41467-017-01481-9](https://doi.org/10.1038/s41467-017-01481-9).
- [41] C. Wang, A. Agrawal, E. Yu, and K. Roy, “Multi-level neuromorphic devices built on emerging ferroic materials: A review,” *Frontiers in Neuroscience*, vol. 15, 2021.
- [42] S. Li, C. Xu, Q. Zou, J. Zhao, Y. Lu, and Y. Xie, “Pinatubo,” in *Proceedings of the 53rd Annual Design Automation Conference on - DAC16*, ACM Press, 2016. DOI: [10.1145/2897937.2898064](https://doi.org/10.1145/2897937.2898064).
- [43] J. Bardeen and W. H. Brattain, “The transistor, a semi-conductor triode,” *Physical Review*, vol. 74, no. 2, p. 230, 1948.
- [44] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, “8t sram cell as a multi-bit dot product engine for beyond von-neumann computing,” *arXiv preprint arXiv:1802.08601*, 2018.
- [45] J. P. Kulkarni, A. Goel, P. Ndai, and K. Roy, “A read-disturb-free, differential sensing 1r/1w port, 8t bitcell array,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 19, no. 9, pp. 1727–1730, Sep. 2011. DOI: [10.1109/tvlsi.2010.2055169](https://doi.org/10.1109/tvlsi.2010.2055169).
- [46] *Predictive Technology Models*. [Online] <http://ptm.asu.edu/>, 2016.
- [47] M. Maymandi-Nejad and M. Sachdev, “A monotonic digitally controlled delay element,” *IEEE Journal of Solid-State Circuits*, vol. 40, no. 11, pp. 2212–2219, 2005.
- [48] “Nios II processor overview,” in *Embedded SoPC Design with Nios II Processor and VHDL Examples*, John Wiley & Sons, Inc., Sep. 2011, pp. 179–188. DOI: [10.1002/9781118146538.ch8](https://doi.org/10.1002/9781118146538.ch8).
- [49] *Advanced Encryption Standard*. [Online] <https://github.com/kokke/tiny-AES-c/>, 2016.

- [50] M. Dworkin, "Recommendation for block cipher modes of operation. methods and techniques," National Inst of Standards and Technology Gaithersburg MD Computer Security Div, Tech. Rep., 2001.
- [51] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.
- [52] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2016. DOI: [10.1109/cvpr.2016.90](https://doi.org/10.1109/cvpr.2016.90).
- [53] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, T. Graepel, and D. Hassabis, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, Jan. 2016. DOI: [10.1038/nature16961](https://doi.org/10.1038/nature16961).
- [54] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.
- [55] A. Krizhevsky *et al.*, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
- [56] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, IEEE, Jun. 2015. DOI: [10.1109/cvpr.2015.7298594](https://doi.org/10.1109/cvpr.2015.7298594).
- [57] M. L. Schneider, C. A. Donnelly, S. E. Russek, B. Baek, M. R. Pufall, P. F. Hopkins, P. D. Dresselhaus, S. P. Benz, and W. H. Rippard, "Ultralow power artificial synapses using nanotextured magnetic josephson junctions," *Science advances*, vol. 4, no. 1, e1701329, 2018.
- [58] G. Srinivasan, A. Sengupta, and K. Roy, "Magnetic tunnel junction enabled all-spin stochastic spiking neural network," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, Mar. 2017, pp. 530–535.
- [59] G. Srinivasan, A. Sengupta, and K. Roy, "Magnetic tunnel junction based long-term short-term stochastic synapse for a spiking neural network with on-chip STDP learning," *Scientific Reports*, vol. 6, no. 1, Jul. 2016. DOI: [10.1038/srep29545](https://doi.org/10.1038/srep29545).

- [60] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European Conference on Computer Vision*, Springer, 2016, pp. 525–542.
- [61] A. Agrawal, A. Jaiswal, C. Lee, and K. Roy, “X-SRAM: Enabling in-memory boolean computations in CMOS static random access memories,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–14, 2018, ISSN: 1549-8328. DOI: [10.1109/TCSI.2018.2848999](https://doi.org/10.1109/TCSI.2018.2848999).
- [62] A. Jaiswal, I. Chakraborty, A. Agrawal, and K. Roy, “8t SRAM cell as a multi-bit dot product engine for beyond von-neumann computing,” *arXiv preprint arXiv:1802.08601*, 2018.
- [63] Y. Chen *et al.*, “Dadiannao: A machine-learning supercomputer,” in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, IEEE Computer Society, 2014, pp. 609–622.
- [64] Y.-H. Chen *et al.*, “Eyeriss: An energy-efficient reconfigurable accelerator for deep convolutional neural networks,” *IEEE Journal of Solid-State Circuits*, vol. 52, no. 1, pp. 127–138, 2017.
- [65] A. Agrawal, A. Ankit, and K. Roy, “SPARE: Spiking neural network acceleration using rom-embedded rams as in-memory-computation primitives,” *IEEE Transactions on Computers*, pp. 1–1, 2018, ISSN: 0018-9340. DOI: [10.1109/TC.2018.2867048](https://doi.org/10.1109/TC.2018.2867048).
- [66] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester, “A 0.3 v vddmin 4+ 2t sram for searching and in-memory computing using 55nm ddc technology,” in *2017 Symposium on VLSI Circuits*, IEEE, 2017, pp. C160–C161.
- [67] S. K. Gonugondla, M. Kang, and N. R. Shanbhag, “A variation-tolerant in-memory machine learning classifier via on-chip training,” *IEEE Journal of Solid-State Circuits*, no. 99, pp. 1–11, 2018.
- [68] C. Eckert, X. Wang, J. Wang, A. Subramaniyan, R. Iyer, D. Sylvester, D. Blaauw, and R. Das, “Neural cache: Bit-serial in-cache acceleration of deep neural networks,” *arXiv preprint arXiv:1805.03718*, 2018.
- [69] A. Ankit *et al.*, “RESPARC: A reconfigurable and energy-efficient architecture with memristive crossbars for deep spiking neural networks,” in *Proceedings of the 54th Annual Design Automation Conference 2017 on - DAC 17*, ACM Press, 2017. DOI: [10.1145/3061639.3062311](https://doi.org/10.1145/3061639.3062311).

- [70] F. Alibart, L. Gao, B. D. Hoskins, and D. B. Strukov, “High precision tuning of state for memristive devices by adaptable variation-tolerant algorithm,” *Nanotechnology*, vol. 23, no. 7, p. 075 201, Jan. 2012. DOI: [10.1088/0957-4484/23/7/075201](https://doi.org/10.1088/0957-4484/23/7/075201).
- [71] A. Chen and M.-R. Lin, “Variability of resistive switching memories and its impact on crossbar array performance,” in *Reliability Physics Symposium (IRPS), 2011 IEEE International*, IEEE, 2011, MY–7.
- [72] A. Biswas and A. P. Chandrakasan, “Conv-ram: An energy-efficient SRAM with embedded convolution computation for low-power cnn-based machine learning applications,” in *Solid-State Circuits Conference-(ISSCC), 2018 IEEE International*, IEEE, 2018, pp. 488–490.
- [73] Z. Jiang, S. Yin, M. Seok, and J.-s. Seo, “Xnor-sram: In-memory computing sram macro for binary/ternary deep neural networks,” in *2018 IEEE Symposium on VLSI Technology*, IEEE, 2018, pp. 173–174.
- [74] A. Agrawal, A. Jaiswal, D. Roy, B. Han, G. Srinivasan, A. Ankit, and K. Roy, “Xcel-ram: Accelerating binary neural networks in high-throughput sram compute arrays,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 66, no. 8, pp. 3064–3076, 2019.
- [75] B. Razavi, “Analogtodigital converter architectures,” in *Principles of Data Conversion System Design*. Wiley-IEEE Press, 1995, pp. 272–, ISBN: 9780470545638. DOI: [10.1109/9780470545638.ch6](https://doi.org/10.1109/9780470545638.ch6).
- [76] “Cacti 6.0: A tool to understand large caches,” [Online],
- [77] S. Jain, A. Ranjan, K. Roy, and A. Raghunathan, “Computing in memory with spin-transfer torque magnetic ram,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 26, no. 3, pp. 470–483, Mar. 2018, ISSN: 1063-8210. DOI: [10.1109/TVLSI.2017.2776954](https://doi.org/10.1109/TVLSI.2017.2776954).
- [78] 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,” in *NIPS-W*, 2017.
- [79] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, *Binarynet.pytorch*, <https://github.com/itayhubara/BinaryNet.pytorch>, 2017.
- [80] A. Krizhevsky and G. Hinton, “Learning multiple layers of features from tiny images,” Citeseer, Tech. Rep., 2009.

- [81] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.
- [82] N. Chatterjee, M. OConnor, D. Lee, D. R. Johnson, S. W. Keckler, M. Rhu, and W. J. Dally, “Architecting an energy-efficient DRAM system for GPUs,” in *2017 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, IEEE, Feb. 2017. DOI: [10.1109/hpca.2017.58](https://doi.org/10.1109/hpca.2017.58).
- [83] M. Horowitz, “1.1 computing’s energy problem (and what we can do about it),” in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, 2014, pp. 10–14.
- [84] A. Agrawal, A. Kosta, S. Kodge, D. E. Kim, and K. Roy, “Cash-ram: Enabling in-memory computations for edge inference using charge accumulation and sharing in standard 8t-sram arrays,” *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 10, no. 3, pp. 295–305, 2020.
- [85] P. N. Whatmough, S. K. Lee, D. Brooks, and G. Wei, “Dnn engine: A 28-nm timing-error tolerant sparse deep neural network processor for iot applications,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2722–2731, 2018.
- [86] S. Jeloka, N. B. Akesh, D. Sylvester, and D. Blaauw, “A 28 nm configurable memory (tcam/bcam/sram) using push-rule 6t bit cell enabling logic-in-memory,” *IEEE Journal of Solid-State Circuits*, vol. 51, no. 4, pp. 1009–1021, 2016.
- [87] Q. Dong, S. Jeloka, M. Saligane, Y. Kim, M. Kawaminami, A. Harada, S. Miyoshi, D. Blaauw, and D. Sylvester, “A 0.3 v vddmin 4+ 2t sram for searching and in-memory computing using 55nm ddc technology,” in *2017 Symposium on VLSI Circuits*, IEEE, 2017, pp. C160–C161.
- [88] Y. Zhang, L. Xu, K. Yang, Q. Dong, S. Jeloka, D. Blaauw, and D. Sylvester, “Re-cryptor: A reconfigurable in-memory cryptographic cortex-m0 processor for iot,” in *2017 Symposium on VLSI Circuits*, 2017, pp. C264–C265.
- [89] M. Kang, M.-S. Keel, N. R. Shanbhag, S. Eilert, and K. Curewitz, “An energy-efficient vlsi architecture for pattern recognition via deep embedding of computation in sram,” in *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, IEEE, 2014, pp. 8326–8330.
- [90] M. Kang, S. K. Gonugondla, A. Patil, and N. R. Shanbhag, “A multi-functional in-memory inference processor using a standard 6t sram array,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 2, pp. 642–655, 2018.

- [91] J. Yang, Y. Kong, Z. Wang, Y. Liu, B. Wang, S. Yin, and L. Shi, "24.4 sandwich-ram: An energy-efficient in-memory bwn architecture with pulse-width modulation," in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, IEEE, 2019, pp. 394–396.
- [92] M. Kang, S. Lim, S. Gonugondla, and N. R. Shanbhag, "An in-memory vlsi architecture for convolutional neural networks," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, vol. 8, no. 3, pp. 494–505, 2018.
- [93] M. Ali, A. Jaiswal, S. Kodge, A. Agrawal, I. Chakraborty, and K. Roy, "Imac: In-memory multi-bit multiplication and accumulation in 6t sram array," *IEEE Transactions on Circuits and Systems I: Regular Papers*, pp. 1–11, 2020.
- [94] X. Si, J.-J. Chen, Y.-N. Tu, W.-H. Huang, J.-H. Wang, Y.-C. Chiu, W.-C. Wei, S.-Y. Wu, X. Sun, R. Liu, *et al.*, "A twin-8t sram computation-in-memory unit-macro for multibit cnn-based ai edge processors," *IEEE Journal of Solid-State Circuits*, vol. 55, no. 1, pp. 189–202, 2019.
- [95] H. Valavi, P. J. Ramadge, E. Nestler, and N. Verma, "A 64-tile 2.4-mb in-memory-computing cnn accelerator employing charge-domain compute," *IEEE Journal of Solid-State Circuits*, vol. 54, no. 6, pp. 1789–1799, 2019.
- [96] H. Jia, H. Valavi, Y. Tang, J. Zhang, and N. Verma, "A programmable heterogeneous microprocessor based on bit-scalable in-memory computing," *IEEE Journal of Solid-State Circuits*, pp. 1–1, 2020.
- [97] Q. Dong, M. E. Sinangil, B. Erbagci, D. Sun, W. Khwa, H. Liao, Y. Wang, and J. Chang, "15.3 a 351tops/w and 372.4gops compute-in-memory sram macro in 7nm finfet cmos for machine-learning applications," in *2020 IEEE International Solid-State Circuits Conference - (ISSCC)*, 2020, pp. 242–244.
- [98] S. Jain, S. K. Gupta, and A. Raghunathan, "Tim-dnn: Ternary in memory accelerator for deep neural networks," *arXiv preprint arXiv:1909.06892*, 2019.
- [99] A. Mishra, E. Nurvitadhi, J. J. Cook, and D. Marr, "Wrpn: Wide reduced-precision networks," *arXiv preprint arXiv:1709.01134*, 2017.
- [100] J. Lee, D. Shin, Y. Kim, and H. J. Yoo, "A 17.5-fj/bit energy-efficient analog sram for mixed-signal processing," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 10, pp. 2714–2723, Oct. 2017, ISSN: 1063-8210. DOI: [10.1109/TVLSI.2017.2664069](https://doi.org/10.1109/TVLSI.2017.2664069).

- [101] M. Hu, H. Li, Y. Chen, Q. Wu, G. S. Rose, and R. W. Linderman, “Memristor crossbar-based neuromorphic computing system: A case study,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 25, no. 10, pp. 1864–1878, Oct. 2014, ISSN: 2162-237X. DOI: [10.1109/TNNLS.2013.2296777](https://doi.org/10.1109/TNNLS.2013.2296777).
- [102] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998. DOI: [10.1109/5.726791](https://doi.org/10.1109/5.726791).
- [103] A. Krizhevsky, “Learning multiple layers of features from tiny images,” Tech. Rep., 2009.
- [104] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.
- [105] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.
- [106] L. Chen, J. Li, Y. Chen, Q. Deng, J. Shen, X. Liang, and L. Jiang, “Accelerator-friendly neural-network training: Learning variations and defects in RRAM crossbar,” in *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, IEEE, Mar. 2017. DOI: [10.23919/date.2017.7926952](https://doi.org/10.23919/date.2017.7926952).
- [107] M. E. Fouda, J. Lee, A. M. Eltawil, and F. Kurdahi, “Overcoming crossbar non-idealities in binary neural networks through learning,” in *Proceedings of the 14th IEEE/ACM International Symposium on Nanoscale Architectures*, ser. NANOARCH ’18, Athens, Greece: ACM, 2018, pp. 31–33, ISBN: 978-1-4503-5815-6. DOI: [10.1145/3232195.3232226](https://doi.org/10.1145/3232195.3232226). [Online]. Available: <http://doi.acm.org/10.1145/3232195.3232226>.
- [108] 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, Oct. 2018. DOI: [10.1109/tetci.2018.2829919](https://doi.org/10.1109/tetci.2018.2829919).
- [109] T. L. Brandon, D. G. Elliott, and B. F. Cockburn, “Using stacked bitlines and hybrid rom cells to form rom and sram-rom with increased storage density,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 53, no. 12, pp. 2595–2605, 2006.
- [110] T. Matsumura, S. Uramoto, and M. Yoshimoto, *Semiconductor memory device usable as static type memory and read-only memory and operating method therefor*, US Patent 5,365,475, Nov. 1994.

- [111] G. M. Ansel, J. S. Hunt, S. Saripella, S. R. Anumula, and A. Srikrishna, *Read only/random access memory architecture and methods for operating same*, US Patent 5,880,999, Mar. 1999.
- [112] S. M. Gold and M. Lamere, *Combining ram and rom into a single memory array*, US Patent 6,438,024, Aug. 2002.
- [113] D. Lee *et al.*, “Area efficient ROM-embedded SRAM cache,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 9, pp. 1583–1595, 2013, ISSN: 10638210. DOI: [10.1109/TVLSI.2012.2217514](https://doi.org/10.1109/TVLSI.2012.2217514).
- [114] D. Lee, X. Fong, and K. Roy, “R-MRAM: A ROM-embedded STT mram cache,” *IEEE Electron Device Letters*, vol. 34, no. 10, pp. 1256–1258, Oct. 2013, ISSN: 0741-3106. DOI: [10.1109/LED.2013.2279137](https://doi.org/10.1109/LED.2013.2279137).
- [115] X. Fong *et al.*, “Embedding read-only memory in spin-transfer torque mram-based on-chip caches,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 3, pp. 992–1002, 2016.
- [116] A. Agrawal and K. Roy, “Recache: Rom-embedded 8-transistor sram caches for efficient neural computing,” in *2018 IEEE International Workshop on Signal Processing Systems (SiPS)*, IEEE, 2018, pp. 19–24.
- [117] A. Agrawal, A. Ankit, and K. Roy, “Spare: Spiking networks acceleration using cmos rom-embedded ram as an in-memory-computation primitive,” *arXiv preprint arXiv:1711.07546*, 2017.
- [118] P. U. Diehl *et al.*, “Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing,” in *Neural Networks (IJCNN), 2015 International Joint Conference on*, IEEE, 2015, pp. 1–8.
- [119] S. Han *et al.*, “Learning both weights and connections for efficient neural network,” in *Advances in Neural Information Processing Systems*, 2015, pp. 1135–1143.
- [120] J. Borghetti *et al.*, “Memristive switches enable stateful logic operations via material implication,” *Nature*, vol. 464, no. 7290, pp. 873–876, 2010.
- [121] A. Ankit, A. Sengupta, and K. Roy, “TraNNsformer: Neural network transformation for memristive crossbar based neuromorphic system design,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, IEEE, Nov. 2017. DOI: [10.1109/iccad.2017.8203823](https://doi.org/10.1109/iccad.2017.8203823).
- [122] C. Clopath *et al.*, “Voltage and spike timing interact in STDP—a unified model,” *Spike-timing dependent plasticity*, p. 294, Jul. 2010.

- [123] T. V. Bliss *et al.*, “A synaptic model of memory: Long-term potentiation in the hippocampus,” *Nature*, vol. 361, no. 6407, p. 31, 1993.
- [124] J. Harrison, T. Kubaska, S. Story, M. S. Labs, and I. Corporation, “The computation of transcendental functions on the ia-64 architecture,” *Intel Technology Journal*, vol. 4, pp. 234–251, 1999.
- [125] F. Akopyan *et al.*, “TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, 2015, ISSN: 02780070. DOI: [10.1109/TCAD.2015.2474396](https://doi.org/10.1109/TCAD.2015.2474396).
- [126] P. Dayan *et al.*, *Theoretical neuroscience*. Cambridge, MA: MIT Press, 2001, vol. 806.
- [127] E. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov. 2003. DOI: [10.1109/tnn.2003.820440](https://doi.org/10.1109/tnn.2003.820440).
- [128] A. L. Hodgkin *et al.*, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, no. 4, pp. 500–544, Aug. 1952. DOI: [10.1113/jphysiol.1952.sp004764](https://doi.org/10.1113/jphysiol.1952.sp004764).
- [129] P. U. Diehl and M. Cook, “Unsupervised learning of digit recognition using spike-timing-dependent plasticity,” *Frontiers in Computational Neuroscience*, vol. 9, Aug. 2015. DOI: [10.3389/fncom.2015.00099](https://doi.org/10.3389/fncom.2015.00099).
- [130] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, “Nvsim: A circuit-level performance, energy, and area model for emerging nonvolatile memory,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 7, pp. 994–1007, Jul. 2012, ISSN: 0278-0070. DOI: [10.1109/TCAD.2012.2185930](https://doi.org/10.1109/TCAD.2012.2185930).
- [131] B. Han, A. Ankit, A. Sengupta, and K. Roy, “Cross-layer design exploration for energy-quality tradeoffs in spiking and non-spiking deep artificial neural networks,” *IEEE Transactions on Multi-Scale Computing Systems*, vol. PP, no. 99, pp. 1–1, 2017. DOI: [10.1109/TMSCS.2017.2737625](https://doi.org/10.1109/TMSCS.2017.2737625).
- [132] D. Goodman, “Brian: A simulator for spiking neural networks in python,” *Frontiers in Neuroinformatics*, vol. 2, 2008. DOI: [10.3389/neuro.11.005.2008](https://doi.org/10.3389/neuro.11.005.2008).
- [133] F. Sun, C. Wang, L. Gong, C. Xu, Y. Zhang, Y. Lu, X. Li, and X. Zhou, “A power-efficient accelerator for convolutional neural networks,” in *2017 IEEE International Conference on Cluster Computing (CLUSTER)*, Sep. 2017, pp. 631–632. DOI: [10.1109/CLUSTER.2017.47](https://doi.org/10.1109/CLUSTER.2017.47).

- [134] M. Davies *et al.*, “Loihi: A neuromorphic manycore processor with on-chip learning,” *IEEE Micro*, vol. 38, no. 1, pp. 82–99, 2018.
- [135] N. Rathi *et al.*, “Diet-snn: Direct input encoding with leakage and threshold optimization in deep spiking neural networks,” *arXiv preprint arXiv:2008.03658*, 2020.
- [136] J. S. P. Giraldo *et al.*, “Laika: A 5uw programmable lstm accelerator for always-on keyword spotting in 65nm cmos,” in *2018-IEEE 44th European Solid State Circuits Conference (ESSCIRC)*, pp. 166–169.
- [137] S. Narayanan *et al.*, “Spinalflow: An architecture and dataflow tailored for spiking neural networks,” in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pp. 349–362.
- [138] A. Agrawal, M. Ali, M. Koo, N. Rathi, A. Jaiswal, and K. Roy, “Impulse: A 65nm digital compute-in-memory macro with fused weights and membrane potential for spike-based sequential learning tasks,” *IEEE Solid-State Circuits Letters*, pp. 1–1, 2021. DOI: [10.1109/LSSC.2021.3092727](https://doi.org/10.1109/LSSC.2021.3092727).
- [139] B. Han *et al.*, “Rmp-snn: Residual membrane potential neuron for enabling deeper high-accuracy and low-latency spiking neural network,” in *Proceedings of the 2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition, CVPR*, pp. 13 558–13 567.
- [140] A. Maas *et al.*, “Learning word vectors for sentiment analysis,” in *Proceedings of the 2011 annual meeting of the association for computational linguistics: Human language technologies*, pp. 142–150.
- [141] M. Liu *et al.*, “A scalable time-based integrate-and-fire neuromorphic core with brain-inspired leak and local lateral inhibition capabilities,” in *2017 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4.
- [142] J. Koo *et al.*, “Area-efficient transposable 6t sram for fast online learning in neuromorphic processors,” in *2019 IEEE Custom Integrated Circuits Conference (CICC)*, pp. 1–4.
- [143] D. Wang *et al.*, “Always-on, sub-300-nw, event-driven spiking neural network based on spike-driven clock-generation and clock- and power-gating for an ultra-low-power intelligent device,” in *2020 IEEE Asian Solid-State Circuits Conference (A-SSCC)*, pp. 1–4.
- [144] S. Jeloka *et al.*, “A configurable tcam/bcam/sram using 28nm push-rule 6t bit cell,” in *2015 IEEE Symposium on VLSI Circuits*, pp. C272–C273.

- [145] J. Wang *et al.*, “14.2 a compute sram with bit-serial integer/floating-point operations for programmable in-memory vector acceleration,” in *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pp. 224–226.
- [146] J.-H. Kim *et al.*, “Z-pim: An energy-efficient sparsity aware processing-in-memory architecture with fully-variable weight precision,” in *2020 IEEE Symposium on VLSI Circuits*, pp. 1–2.

VITA

Amogh Agrawal received the B.Tech. degree in Electrical Engineering from the Indian Institute of Technology, Ropar, India, in 2016. He joined the Nanoelectronics Research Laboratory in 2016 for pursuing his Ph.D. degree with the School of Electrical and Computer Engineering, Purdue University, under the guidance of Prof. Kaushik Roy. He was a Research Intern with the University of Ulm, Germany, in 2015, under the DAAD (German Academic Exchange Service) Fellowship. He was a Technology Development Intern with GlobalFoundries, Malta, NY, USA, in 2018, and an Engineering Intern with Rambus Inc., Sunnyvale, CA, USA, in 2019. His research interests include enabling in-memory computations for neuromorphic systems using CMOS and beyond-CMOS memories. He was a recipient of the Director’s Gold Medal for his all-round performance, and the Institute Silver Medal for his academic achievements at IIT Ropar. Since 2016, he has been a recipient of the Andrews Fellowship from Purdue University.

PUBLICATIONS

1. T. Sharma, C. Wang, A. Agrawal, and K. Roy, "Enabling Robust SOT-MTJ Crossbars for Machine Learning using Sparsity-Aware Device-Circuit Co-design." In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '21).
2. A. Agrawal, M. Ali, M. Koo, N. Rathi, A. Jaiswal and K. Roy, "IMPULSE: A 65nm Digital Compute-in-Memory Macro with Fused Weights and Membrane Potential for Spike-based Sequential Learning Tasks," in IEEE Solid-State Circuits Letters, doi: 10.1109/LSSC.2021.3092727.
3. M. Ali, I. Chakraborty, U. Saxena, A. Agrawal, A. Ankit and K. Roy, "A 35.5-127.2 TOPS/W Dynamic Sparsity-Aware Reconfigurable-Precision Compute-in-Memory SRAM Macro for Machine Learning," in IEEE Solid-State Circuits Letters, vol. 4, pp. 129-132, 2021, doi: 10.1109/LSSC.2021.3093354.
4. A. Agrawal, A. P. Jacob, "Apparatus and method for in-memory binary convolution for accelerating deep binary neural networks based on a non-volatile memory structure." U.S. Patent No. 10,997,498. 4 May 2021.
5. A. Agrawal, C. Wang, T. Sharma and K. Roy, "Magnetoresistive Circuits and Systems: Embedded Non-Volatile Memory to Crossbar Arrays," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 68, no. 6, pp. 2281-2294, June 2021, doi: 10.1109/TCSI.2021.3069682.
6. A. Agrawal, D. Roy, U. Saxena and K. Roy, "Embracing Stochasticity to Enable Neuromorphic Computing at the Edge," in IEEE Design & Test, doi: 10.1109/MDAT.2021.3051399.
7. C. Wang, A. Agrawal, E. Yu, K. Roy, "Multi-Level Neuromorphic Devices Built on Emerging Ferroic Materials: A Review." in Front Neurosci. 2021;15:661667. Published 2021 Apr 28. doi:10.3389/fnins.2021.661667

8. A. Jaiswal, A. Agrawal, K. Roy, I. Chakraborty, "Multi-bit dot product engine." U.S. Patent No. 10,825,510. 3 Nov. 2020.
9. A. Jaiswal, A. Agrawal, K. Roy, "Memory device having in-situ in-memory stateful vector logic operation." U.S. Patent No. 10,802,827. 13 Oct. 2020.
10. A. Ankit, I. Chakraborty, A. Agrawal, M. Ali and K. Roy, "Circuits and Architectures for In-Memory Computing-Based Machine Learning Accelerators," in IEEE Micro, vol. 40, no. 6, pp. 8-22, 1 Nov.-Dec. 2020, doi: 10.1109/MM.2020.3025863.
11. M. Ali, A. Agrawal, and K. Roy, "RAMANN: in-SRAM differentiable memory computations for memory-augmented neural networks." In Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED '20). Association for Computing Machinery, New York, NY, USA, 61–66. DOI: <https://doi.org/10.1145/3370748.3406574>
12. A. Agrawal, A. Kosta, S. Kodge, D. E. Kim and K. Roy, "CASH-RAM: Enabling In-Memory Computations for Edge Inference Using Charge Accumulation and Sharing in Standard 8T-SRAM Arrays," in IEEE Journal on Emerging and Selected Topics in Circuits and Systems, vol. 10, no. 3, pp. 295-305, Sept. 2020, doi: 10.1109/JET-CAS.2020.3014250.
13. A. P. Jacob, and A. Agrawal, "Resistive nonvolatile memory structure employing a statistical sensing scheme and method." U.S. Patent No. 10,726,896. 28 Jul. 2020.
14. A. Jaiswal, A. Agrawal, M. F. Ali, S. Sharmin and K. Roy, "i-SRAM: Interleaved Wordlines for Vector Boolean Operations Using SRAMs," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 12, pp. 4651-4659, Dec. 2020, doi: 10.1109/TCSI.2020.3005783.
15. A. Jaiswal, A. Agrawal, P. Panda and K. Roy, "Neural Computing With Magnetoelectric Domain-Wall-Based Neurosynaptic Devices," in IEEE Transactions on Magnetics, vol. 57, no. 2, pp. 1-9, Feb. 2021, Art no. 4300209, doi: 10.1109/TMAG.2020.3010712.

16. K. Roy, I. Chakraborty, M. Ali, A. Ankit and A. Agrawal, "In-Memory Computing in Emerging Memory Technologies for Machine Learning: An Overview," 2020 57th ACM/IEEE Design Automation Conference (DAC), 2020, pp. 1-6, doi: 10.1109/DAC18072.2020.9218505.
17. I. Chakraborty et al., "Resistive Crossbars as Approximate Hardware Building Blocks for Machine Learning: Opportunities and Challenges," in *Proceedings of the IEEE*, vol. 108, no. 12, pp. 2276-2310, Dec. 2020, doi: 10.1109/JPROC.2020.3003007.
18. A. Agrawal, A. P. Jacob, "Neuromorphic memory device." U.S. Patent No. 10,672,465. 2 Jun. 2020.
19. A. P. Jacob, A. Agrawal, and B. C. Paul, "Resistive nonvolatile memory cells with shared access transistors." U.S. Patent No. 10,665,281. 26 May 2020.
20. A. Agrawal et al., "Revisiting Stochastic Computing in the Era of Nanoscale Non-volatile Technologies," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 12, pp. 2481-2494, Dec. 2020, doi: 10.1109/TVLSI.2020.2991679.
21. M. Ali, A. Jaiswal, S. Kodge, A. Agrawal, I. Chakraborty and K. Roy, "IMAC: In-Memory Multi-Bit Multiplication and ACcumulation in 6T SRAM Array," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 67, no. 8, pp. 2521-2531, Aug. 2020, doi: 10.1109/TCSI.2020.2981901.
22. A. Jaiswal, R. Andrawis, A. Agrawal and K. Roy, "Functional Read Enabling In-Memory Computations in 1Transistor—1Resistor Memory Arrays," in *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 67, no. 12, pp. 3347-3351, Dec. 2020, doi: 10.1109/TCSII.2020.2975658.
23. I. Chakraborty, A. Agrawal, A. Jaiswal, G. Srinivasan and K. Roy, "In situ unsupervised learning using stochastic switching in magneto-electric magnetic tunnel junctions", in *Phil. Trans. R. Soc. A*. 378: 20190157.20190157
24. A. Jaiswal, I. Chakraborty, A. Agrawal and K. Roy, "8T SRAM Cell as a Multibit Dot-Product Engine for Beyond Von Neumann Computing," in *IEEE Transactions on*

- Very Large Scale Integration (VLSI) Systems, vol. 27, no. 11, pp. 2556-2567, Nov. 2019, doi: 10.1109/TVLSI.2019.2929245.
25. A. Jaiswal, A. Agrawal, I. Chakraborty, D. Roy and K. Roy, "On Robustness of Spin-Orbit-Torque Based Stochastic Sigmoid Neurons for Spiking Neural Networks," 2019 International Joint Conference on Neural Networks (IJCNN), 2019, pp. 1-6, doi: 10.1109/IJCNN.2019.8851780.
 26. A. Agrawal, C. Lee, and K. Roy, "X-CHANGR: Changing memristive crossbar mapping for mitigating line-resistance induced accuracy degradation in deep neural networks." arXiv preprint arXiv:1907.00285 (2019).
 27. A. Jaiswal, A. Agrawal, I. Chakraborty, M. Ali, and K. Roy, "Digital and Analog-Mixed-Signal In-Memory Processing in CMOS SRAM." In Proceedings of the 2019 on Great Lakes Symposium on VLSI, pp. 371-371. 2019.
 28. A. Agrawal et al., "Xcel-RAM: Accelerating Binary Neural Networks in High-Throughput SRAM Compute Arrays," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 66, no. 8, pp. 3064-3076, Aug. 2019, doi: 10.1109/TCSI.2019.2907488.
 29. A. Agrawal and K. Roy, "Mimicking Leaky-Integrate-Fire Spiking Neuron Using Automation of Domain Walls for Energy-Efficient Brain-Inspired Computing," in IEEE Transactions on Magnetics, vol. 55, no. 1, pp. 1-7, Jan. 2019, Art no. 1400107, doi: 10.1109/TMAG.2018.2882164.
 30. A. Agrawal and K. Roy, "RECache: ROM-Embedded 8-Transistor SRAM Caches for Efficient Neural Computing," 2018 IEEE International Workshop on Signal Processing Systems (SiPS), 2018, pp. 19-24, doi: 10.1109/SiPS.2018.8598290.
 31. A. Agrawal, A. Ankit and K. Roy, "SPARE: Spiking Neural Network Acceleration Using ROM-Embedded RAMs as In-Memory-Computation Primitives," in IEEE Transactions on Computers, vol. 68, no. 8, pp. 1190-1200, 1 Aug. 2019, doi: 10.1109/TC.2018.2867048.

- 32. A. Agrawal, A. Jaiswal, C. Lee and K. Roy, "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4219-4232, Dec. 2018, doi: 10.1109/TCSI.2018.2848999.
- 33. I. Chakraborty, A. Agrawal and K. Roy, "Design of a Low-Voltage Analog-to-Digital Converter Using Voltage-Controlled Stochastic Switching of Low Barrier Nanomagnets," in IEEE Magnetics Letters, vol. 9, pp. 1-5, 2018, Art no. 3103905, doi: 10.1109/LMAG.2018.2839097.
- 34. A. Jaiswal, A. Agrawal, and K. Roy, "In-situ, In-Memory Stateful Vector Logic Operations based on Voltage Controlled Magnetic Anisotropy." Sci Rep 8, 5738 (2018). <https://doi.org/10.1038/s41598-018-23886-2>.
- 35. A. Jaiswal, A. Agrawal and K. Roy, "Robust and Cascadable Nonvolatile Magnetoelectric Majority Logic," in IEEE Transactions on Electron Devices, vol. 64, no. 12, pp. 5209-5216, Dec. 2017, doi: 10.1109/TED.2017.2766570.