MID SEMESTER
PROJECT REPORT

ON

‘IN MEMORY COMPUTING’

BY

HARSHIV CHANDRA   2020A7PS0085U   COMPUTER SCIENCE

Submitted in partial fulfillment of
EEE F491 SPECIAL PROJECT

Under the supervision of

Prof. M. B. Srinivas
Professor
Electrical & Electronics Engineering Department

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DUBAI CAMPUS, DUBAI, U.A.E.

1st Semester
Academic Year 2023-24

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI
DUBAI CAMPUS, DUBAI, U.A.E.

CERTIFICATE

This is to certify that the project entitled, '**In Memory Computing'** and submitted by **Harshiv Chandra**, ID No. **2020A7PS0085U** in partial fulfillment of the requirement of EEE F491 Special Project embodies the work done by him/her under my supervision.

Date:

Signature of the Supervisor

Name: M. B. Srinivas

Designation: Professor

BITS, Pilani – Dubai Campus
Dubai International Academic City (DIAC)
Dubai, UAE

Course Name: Special Project

Course No: EEE F491

Duration: 2 months

Date of Start: 28th August, 2023

Date of Submission: 14th November, 2023

Title of Report: In Memory Computing

Name: Harshiv Chandra

ID Number: 2020A7PS0085U

Discipline: Computer Science

Name of Project Supervisor: Prof. M. B. Srinivas

Keywords: memristor, resistive RAM, in memory computing, crossbar array, majority inverted graph

Research Area: in memory computing

Abstract: As computational needs for applications like machine learning and neural networks tend to increase, Silicon MOSFET scaling alone cannot catch up. Silicon MOSFET scaling is hitting its limits, and researchers are working on newer approaches to computing for overcoming this bottleneck. There are two directions in this regard – device level approaches like replacing the Silicon with MOS2 or 3D stacking, or the whole von-Neumann architecture can be overhauled. In this project, a novel majority-minority based memristor primitive is to be implemented. This is to be compared against the existing primitives like MAGIC and IMPLY (ref. Kvatinsky group Technion). An extensive literature review is to be performed as a starting point. Voltage sensitivity and parasitic effects are to be considered. The primitive is then mapped on a memristor-based crossbar array using an algorithm. Delay, power, and area metrics of the primitives are compared.

Harshiv

Signature of the Student	**Signature of Faculty**
Date: 14/11/2023	**Date:**

ACKNOWLEDGEMENT

At the outset, I would like to thank my supervisor Prof. M. B. Srinivas who gave me this wonderful opportunity to work on the project which helped me in understanding memristor-based computing and get enriched with knowledge of unknown dimensions. In the process, I explored and worked on various RRAM based logic primitives and various architectures associated with it. This also helped me revisit CMOS VLSI design, and draw parallelisms to it.

I would like to thank the director of BITS Pilani, Dubai campus Prof. Srinivas Madapusi who has always been keen on creating an environment for placing our university at the helm of cutting-edge research.

I would also like to thank my parents and friends who have helped me directly to complete this project. Any amount of thanks will be less to acknowledge the websites, journal papers, conference papers, and books from which I have used resources in our project.

Harshiv Chandra

2020A7PS0085U

CONTENTS

LIST OF FIGURES

CHAPTER 1.

INTRODUCTION

1.1. CMOS Technology

CMOS stands for Complementary Metal Oxide Semiconductor FET technology. Before delving into it, it is imperative to talk about a MOSFET which stands for Metal Oxide Semiconductor Field Effect Transistor. It was first demonstrated by M. M. (John) Atalla and Dawon Kahng at Bell Labs in 1959 [1]. Fig. 1 demonstrates the first MOSFET from the patent filed.

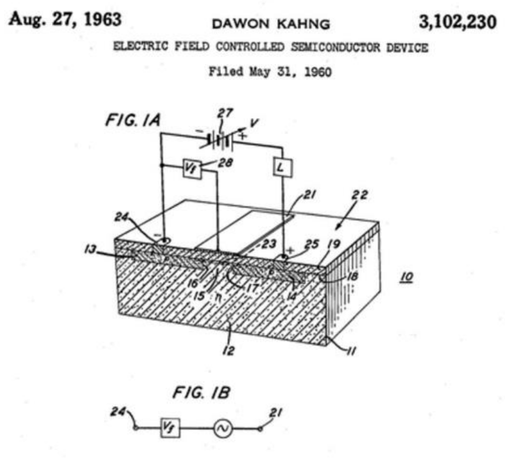


Fig. 1. Diagram of the first MOSFET

MOSFETs are voltage-controlled devices, and applying a voltage at the gate allows current to flow from source to drain. This property of a MOSFET is exploited to build logic circuits. MOSETs could be modelled as ideal switches, and hence logic circuits could be design based on it. Many paradigms were tired, and industry stuck with CMOS technology – using pull up (PMOS) and pull down (NMOS) networks to design logic circuits. This was complemented with advances in lithography technology which allowed to fabricate transistors akin to printing them on the Silicon. Lithography technology is expensive, and a single machine costs around US$ 150 million dollars. Fig. 2 shows TWINSCAN NXE:3600D, ASML's most advanced lithography machine which costs upwards of US$ 300 million [2].

Fig. 2. ASML's TWINSCAN NXE:3600D Lithography Machine

With advances in lithography, MOSFETs scaled down making the integrated circuits faster, cheaper, and power efficient. Number of transistors in a microprocessor scaled from a few thousands to around 10 million by 1996. This remarkable scaling has been summed up nicely by the following quote of Mark Stephens – “*If the automobile had followed the same development cycle as the computer, a Rolls-Royce would today cost $100, get a million miles per gallon, and explode once a year, killing everyone inside.* [3]”

1.2. The Moore’s Law

Gordon Moore, the cofounder of Intel, predicted in 1965 that number of transistors in an integrated circuit would double every year [4]. He later revised his prediction in 1975 where he stated that the number of transistors on an integrated circuit would double every two years [5]. This prediction has been true since 1975, and has been held as the Moore’s law which has been driving scaling advancements in the semiconductor industry. Fig. 3 illustrates how Moore’s law has been performing since the invention of MOS based integrated circuit.

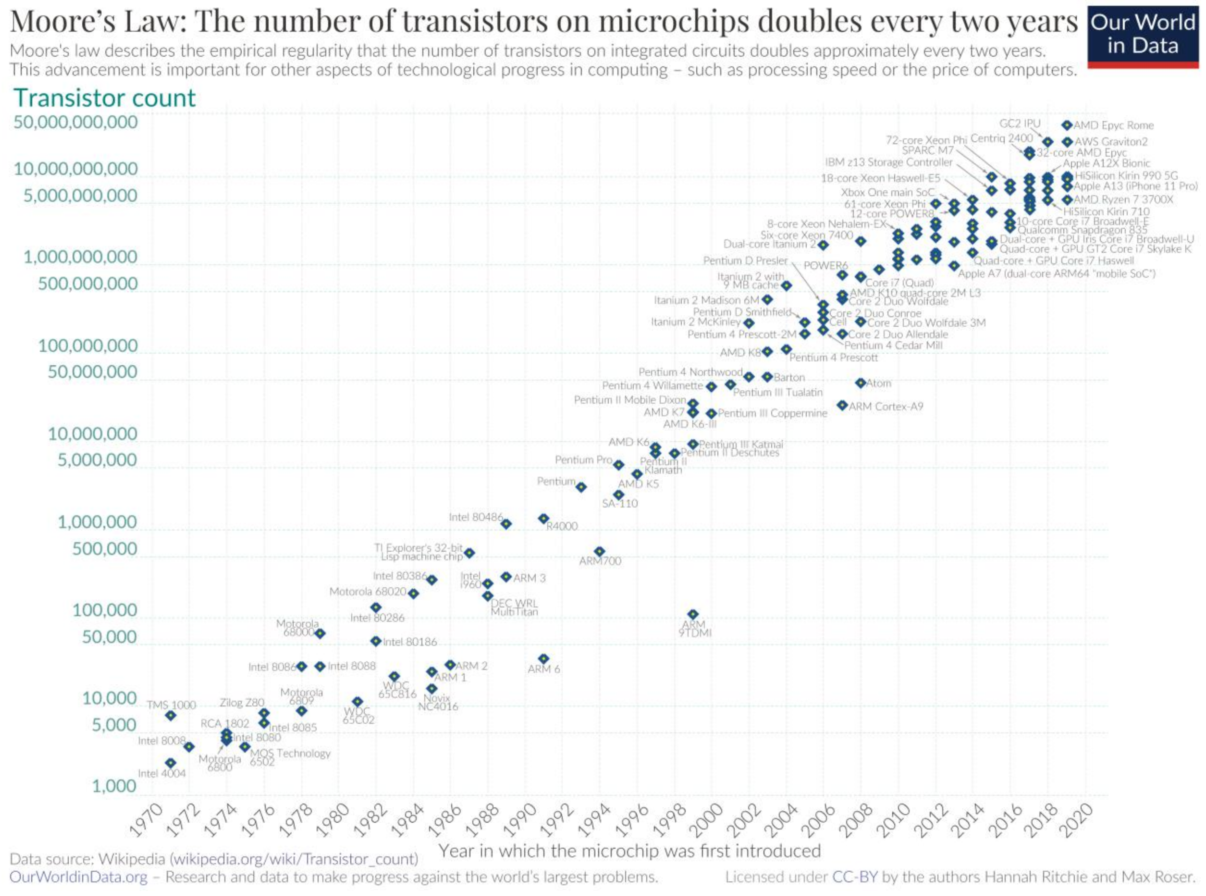


Fig. 3. Transistor count over the years (y-axis in semi-log scale) [6]

Moore's law is no stranger to dangers – a similar problem occurred in 1996, popularly known as the short channel effect. This was caused by scaling the MOSFET, which resulted in a shorter and thinner channel (around 1.2 nm in 1996) which gave the gate less control over the transistor. This in turn resulted in the channel being shorted despite the fact that the gate voltage was off. This led to incorrect logic computations and more power consumption as the MOSFET happened to be ON on the time. The industry envisioned that transistors would not scale beyond 25 nm which was in the year 2002. Defense Advanced Research Projects Agency (DARPA) stepped in with its proposal for research on sub-25 nm devices in 1997 [7]. Hu Chenming's proposal was accepted, and his team developed the first FinFET in 2001. FinFET raises the source and drain of a planar MOSFET for the gate to have better control over the channel (fig. 4). The semiconductor industry did not adopt it soon. Intel came up with high-K metal gates (HfO2) for its 28 nm process node. This was eventually adopted by Intel in its Tri-gate technology. Present ways to keep the Moore's law alive includes Gate-All-Around FETs (GAAFET) (fig. 4), where the gate wraps the channel on all the four sides. This needs a special process to fabricate, which is known as atomic layer deposition.

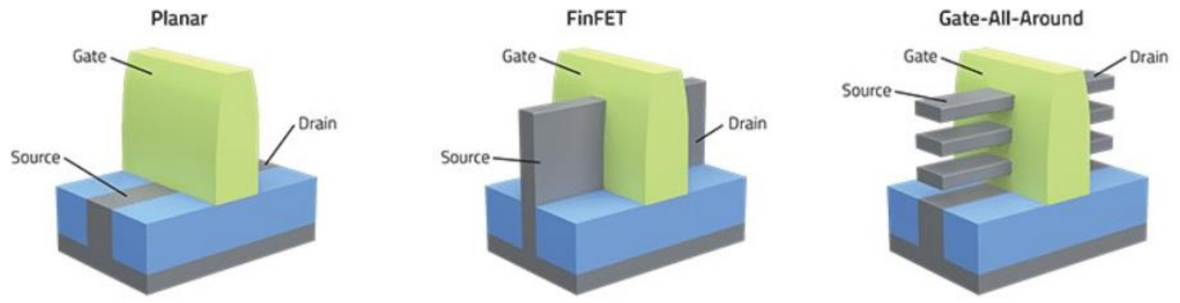


Fig. 4. Planar MOSFET, FinFET, and the Gate-All-Around FET

This creative desperation points to replacing either the computational architectures for processing, or replacing the device, or both. The next section talks about the contemporary contenders to the present Silicon CMOS technology.

1.3. Beyond CMOS Devices and Architectures

As aforementioned, the Silicon CMOS technology seems to be on the verge of obsoletion for newer process nodes (and/or faster computations), there seem to be a few contenders to it. They can be broadly categorized as follows:

1.3.1. Device level

These paradigms try to replace the device (which is a version of the planar Silicon MOSFET), with a different device. These can include MOSFETs made out of some other material – Molybdenum Disulphide (MoS2) transistors [8] (fig. 5) and Carbon nanotube FETs [9] (fig. 6). A totally new device is a resistive RAM which is the subject of this project. For the sake of completeness, it is a device which has a non-volatile variable resistance which can be exploited for computations.

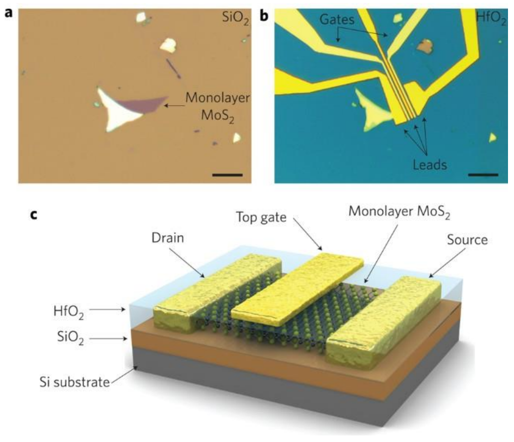


Fig. 5 Single layer MoS2 transistor

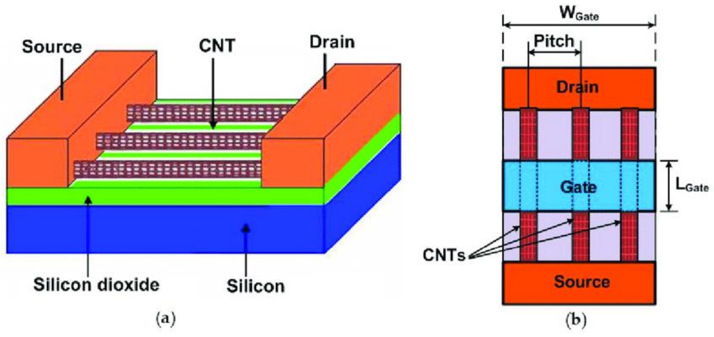


Fig. 6. Carbon nanotube Field Effect Transistor

1.3.2. Architecture level

A prominent example of this neuromorphic computing, which is essentially replicating the spiking structures in our brain, the smallest unit being a synapse and a neuron. This work was pioneered by Prof. Craver Mead in the 1980s [10] when he developed the concept of an electronic neuron and synapse. This work has been taken further, and researchers are employing techniques to implement neuromorphic CMOS circuits beyond 45 nm process node. This has given way to spiking neural networks (SNN) which brings us to the next section. Another area is in-memory computing using traditional using traditional CMOS structures. [11] demonstrates a 6T SRAM cell using Silicon CMOS which incorporates in memory computing.

1.3.3. Device and Architecture level

This area is totally radical, and has been the subject of research for only a decade. Here, memristor (particulary RRAM) based spiking neural networks are the main focus, where SNNs are implemented using resistive RAMs to perform arithmetic computations (which extends to other tasks). A 38-core chip with 3 million RRAM cells has been demonstrated in (fig. 7) [12] with high accuracies in certain benchmarks. This shows how promising the technology is, and what it means for the future of computing.

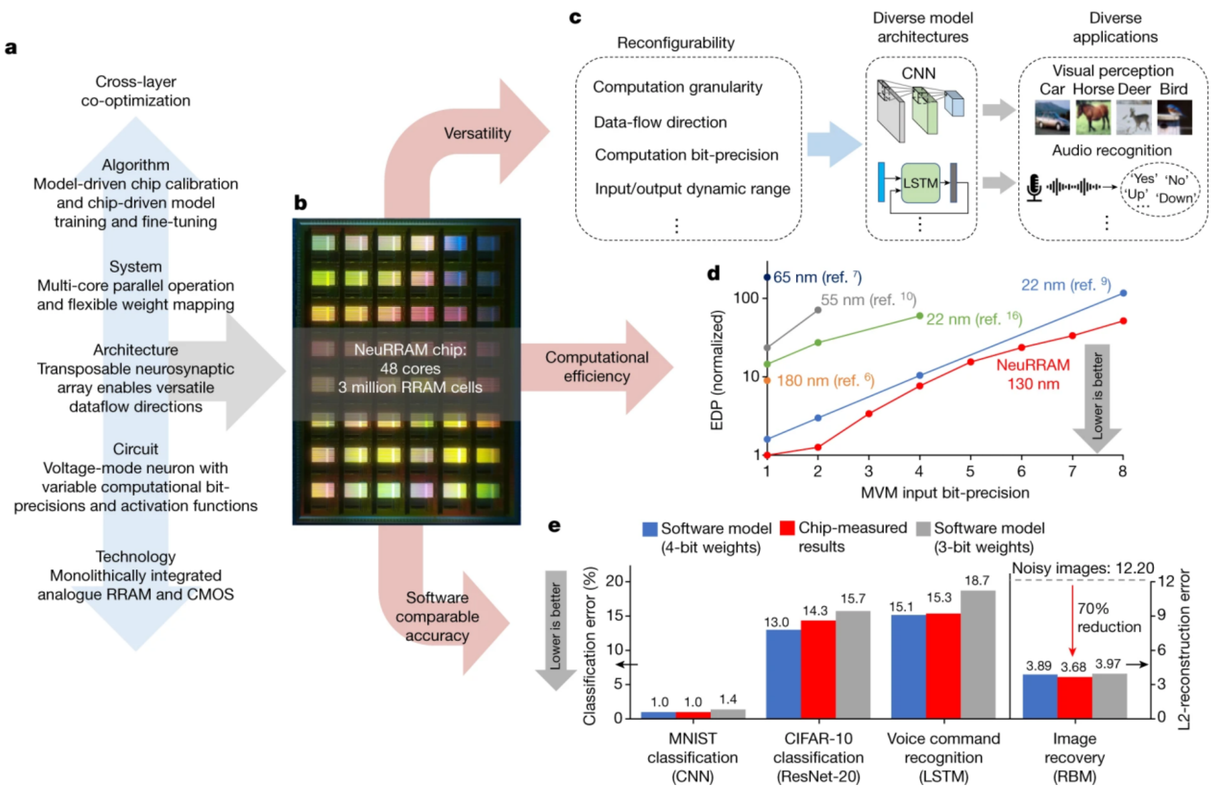


Fig. 7. Overview of the NeuRRAM chip with 48 cores and 3 million RRAM cells

CHAPTER 2. LITERATURE REVIEW

Several methods on majority logic mapping have been proposed in literature. [13] discusses the efficiency of MAJORITY+NOT (which makes it functionally complete) logic [14] over the other conventional logic primitives, like IMPLY [15] [16], IMPLY (semi-parallel) [17], ORNOR [18], NOR [19] [20], and NAND [21], by making a comparison of the latency obtained by performing In-memory computations with 1-bit full adder and 8-bit full adder. In comparison with NAND/NOR logic primitive, the MAJORITY+NOT logic can reduce the logic levels of 1-bit full adder by 33-43%. For example, implementation of a one-bit full adder requires 10 cycles by each of the NAND and NOR logic primitives, whereas MAJORITY logic achieves the same by 6 cycles in a 1T-1R array [14]. Also, MAJORITY+NOT achieves to implement an 8-bit full adder in 7 levels and 19 cycles [22].

[23] opens up a novel gateway for logic optimization and synthesis by proposing a delay-oriented optimization technique. It shows that MIGs reduce the logical depth by 18.6% than AIGs and by 23.7% than decomposed BDDs. The delay, area, and power generated by MIG flow are respectively 22%, 14%, and 11% smaller than that achieved by the best existing commercial synthesis tool.

[13] also makes a comparison between the two non-stateful logics: V-R [24][25][26] and R-V [10][11]. In V-R logic [24][25][26], the conventional way of row/column decoding becomes complicated as modification in selecting row and applying inputs

is required during the memory operation and majority operation respectively. Whereas, the R-V logic [14][22] is implemented by selecting three rows during majority operation which is achieved by interleaving decoders. In [13], the logical operations (logic 0 and logic 1) are performed by sensing the effective resistance as <= 4.8 K-ohm and >=8.7 K-ohm. The variability in states is a major issue in RRAM. Hence, to counter this drawback, [14] uses a current-mode SA and [22] uses a time-based SA.

[14] shows a way to implement majority logic in terms of majority current $I_{MAJ.}$ For the SA to read a single cell and to compute the majority logic, the applied voltage V_R is taken to be 0.3 V, and the resulting currents (I_{HRS} and I_{LRS}) are 4.5 micro-Amp and 45 micro-Amp respectively, which is achieved without affecting the resistive states of IHP's cells whose SET and RESET voltages are 0.9 V and -1.1 V respectively. The SA was made to distinguish between the resulting currents as <= 18 micro-Amp and >= 31.5 micro-Amp, and for greater margin clarity the I_{REF} is taken to be 24.75 micro-Amp.

[27] presents a novel architecture known as 'Wallace Tree multiplier architecture' to counteract the inefficiency in terms of power and latency caused by the existing in-memory multipliers that require $O(n^2)$ cycles. In some recent research works [28][29], majority gates have been used to synthesize parallel-prefix adders. In a 45 nm CMOS technology, multiplication is carried out to find the energies of READ (Majority) and WRITE operations [27], which are found to be E_{READ} = 2.2 pJ/operation for a READ cycle duration of 20 ns and E_{WRITE} = 8.2 pJ/bit for a WRITE cycle duration of 50 ns. For 8*8 multiplier, the total number of READ and WRITE operations are 283 and 556 respectively, and hence the total energies for the respective operations are 283 * 2.2 pJ/opration = 622 pJ and 556 *8.2 pJ/bit = 5181 pJ. The 8*8 Also, one crucial advantage of Wallace Tree architecture using majority logic [27] is evident by the increase in the logic levels from 12 for a 4*4 multiplier to 19 for a 8*8 multiplier.

CHAPTER 3.

MAGIC and IMPLY PRIMITIVE USING MEMRISTORS

3.1. IMPLY Primitive

IMPLY or material implication [16] is a memristor based primitive developed by Kvatinsky group at Technion. This is a stateful logic primitive with two memristors – the initial logic values are initialised as logic states of the two memristors, and then after the first cycle, the output is stored as a state in one of the memristors. The disadvantage of this primitive is that the inputs are destroyed, and this requires as additional resistor. The Cadence Virtuoso implementation of IMPLY NOR logic is as follows:

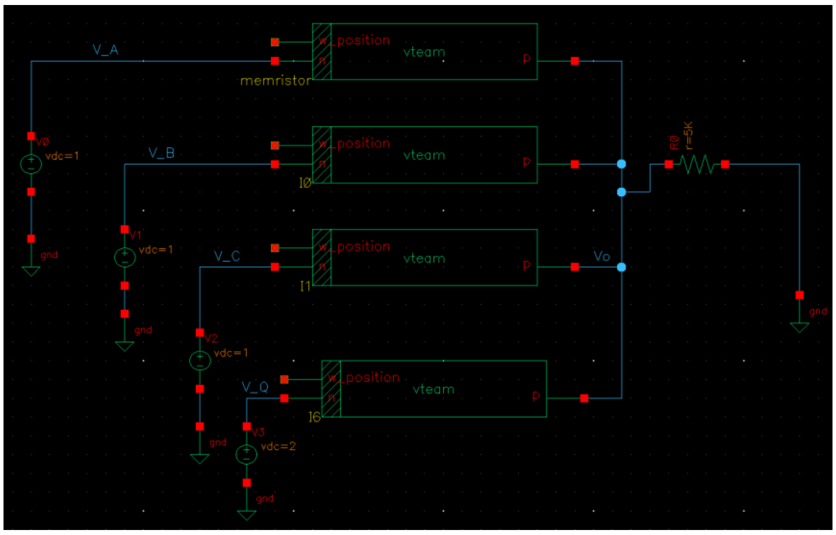


Fig. 8. Implementation of IMPLY in Cadence Virtuoso

The resistances are varied for all the eight input variables using the CDF parameters, and the output is calculated as the resistance across the memristor 'I6'. For the CDF variable states as R_on, the output is show in fig. 9.

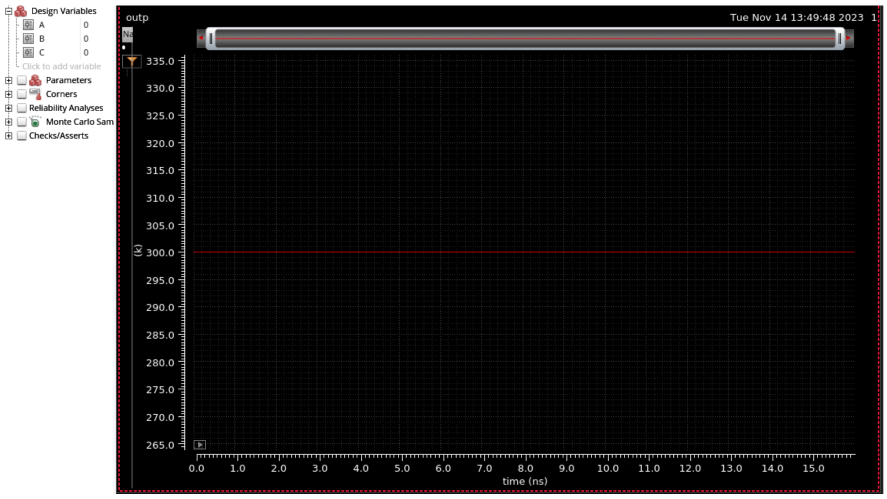


Fig. 9. Output of IMPLY NOR

The VTEAM model parameters are shown below:

Parameter	Value	Parameter	Value
R_{on}	$1.0k\Omega$	x_{on}	$0nm$
R_{off}	$300k\Omega$	x_{off}	$3nm$
V_{on}	$-1.5V$	α_{on}	4
V_{off}	$0.3V$	α_{off}	4
K_{on}	$-285.6ms^{-1}$	k_{off}	$0.09ms^{-1}$

3.2. MAGIC Primitive

MAGIC or memristor aided logic [19] is a popular stateful memristor primitive which has an additional memristor to store the output. In this way, the inputs do not get destroyed as the output is stored in a separate memristor. The Cadence Virtuoso implementation of MAGIC NOR is shown in fig. 10.

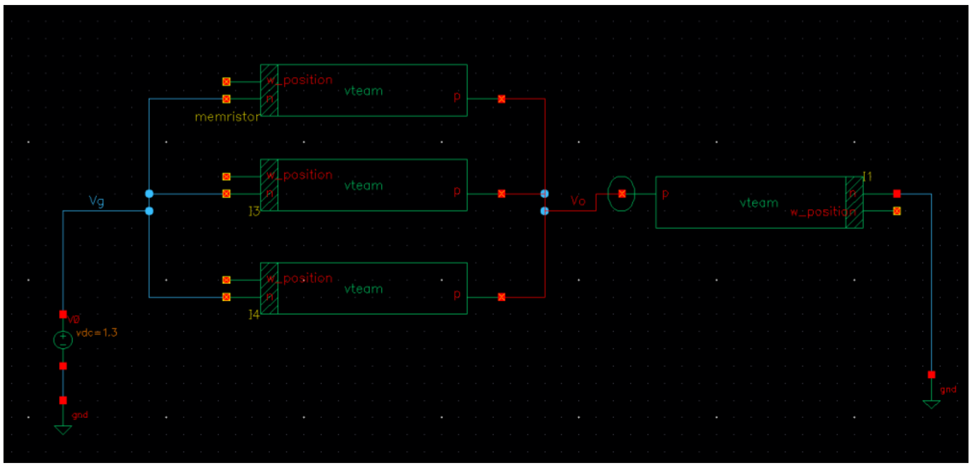
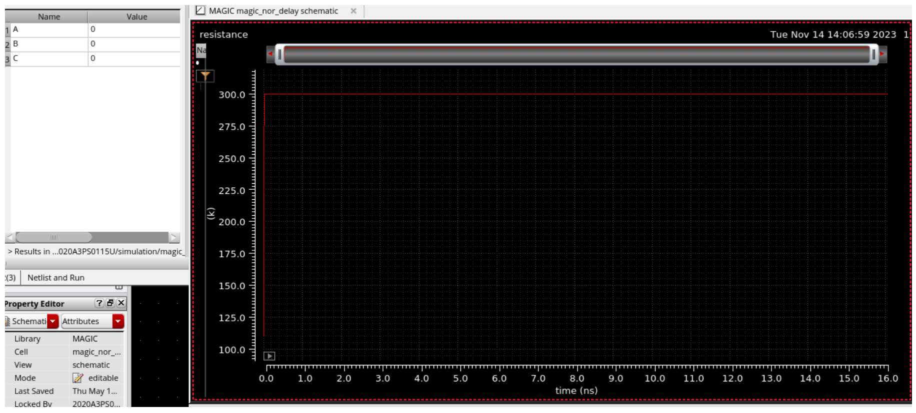


Fig. 10. Implementation of MAGIC in Cadence Virtuoso

The output is shown in fig. 11 for the three input variables as R_on.



	Name	Value
1	A	0
2	B	0
3	C	0

> Results in ...020A3PS0115U/simulation/magic_

Netlist and Run

Property Editor

Library	MAGIC
Cell	magic_nor_...
View	schematic
Mode	editable
Last Saved	Thu May 1...
Locked By	2020A3PS0...

Fig. 11. Output of MAGIC NOR

CHAPTER 4.

MAJORITY/MINORITY UNIVERSAL LOGIC MAPPING FOR MEMRISTOR CROSSBAR ARRAY (MUGIC)

4.1. MUGIC Primitive

MUGIC is a memristor based logic primitive which is based on majority/minority logic, and it can be used to realize any n- input logic gate by mapping them onto the memristor based crossbar array.

A Majority logic gate is an n- input logic gate whose output is based on the majority of the inputs. For a three-input Majority gate, the Boolean expression is given as:

$$MAJ(A, B, C) = AB + BC + CA$$

A schematic of a three-input Majority gate implemented in cadence virtuoso is shown in fig. 12 along with its SPICE simulation result in fig. 13.

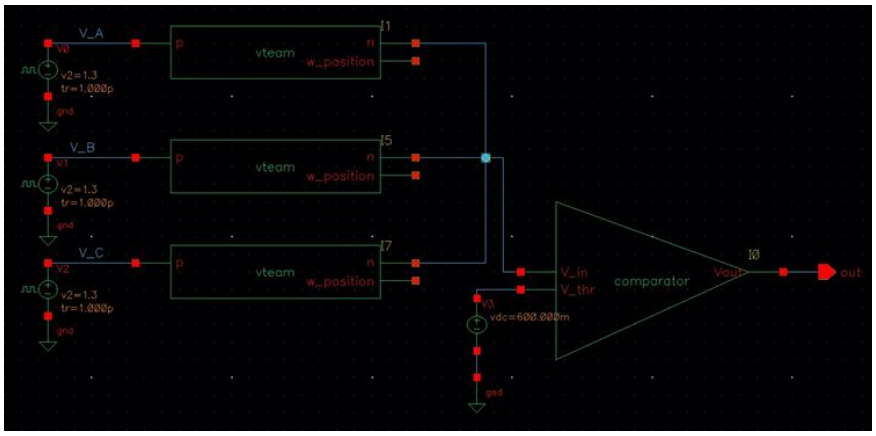


Fig. 12. Three-input Majority gate implemented in cadence virtuoso

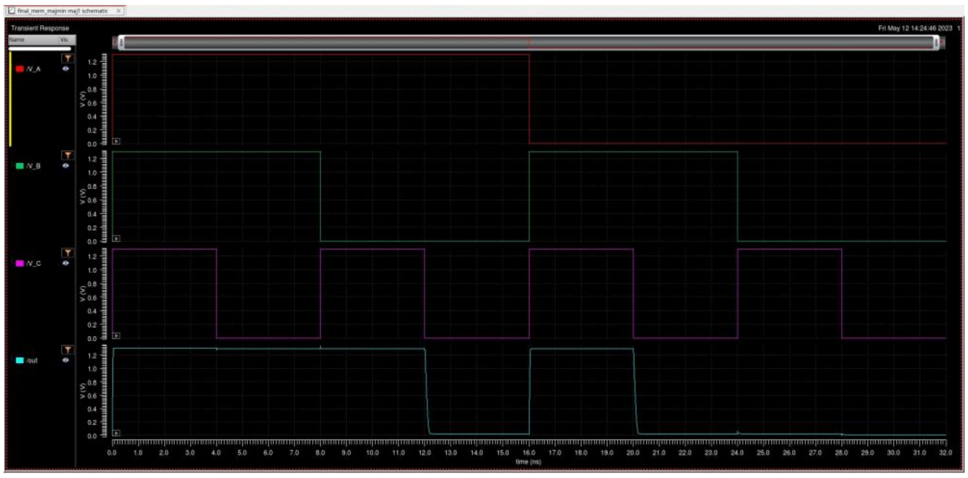


Fig. 13. SPICE simulation result of a three-input Majority gate

Similarly, a Minority logic gate is an n-input logic gate whose output is based on the minority of the inputs. In other words, a Minority gate is just the inversion of a Majority gate.

A schematic of a three-input Minority gate implemented in cadence virtuoso is shown in fig. 14 along with its SPICE simulation result in fig. 15.

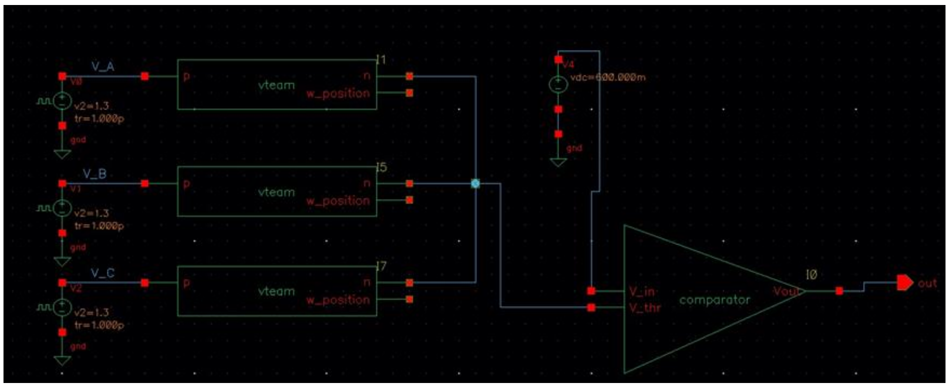


Fig. 14. Three-input Minority gate implemented in cadence virtuoso

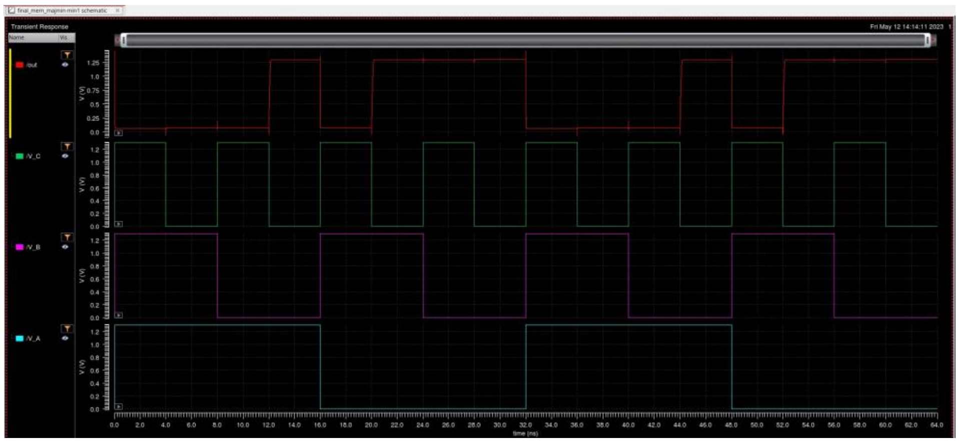


Fig. 15. SPICE simulation result of a three-input Minority gate

The majority/minority logic is based on the traditional threshold logic. In Majority logic, the input is fed to the positive terminal of the comparator, whereas in Minority logic, the input is fed to the negative terminal of the comparator.

In the proposed design of MUGIC, the threshold voltage V_{thr} of the comparator is considered to be between $V_{high}/3$ and $2V_{high}/3$ with the other design consideration given below.

Design Variable	Design Consideration
$\boldsymbol{V_{high}}$	$V_{high} > V_{off}$
$\boldsymbol{V_{low}}$	$V_{on} < V_{low} < V_{off}$
$\boldsymbol{V_{thr}}$	$V_{high}/3 < V_{thr} < 2V_{high}/3$

In the proposed design consideration, the V_{high} is taken as 1.3 V and the V_{low} is taken as 0 V.

4.2. Energy Comparison

A comparative study has been performed on the energy dissipation of a single Majority/Minority gate, a MAGIC NOR gate, and an IMPLY NOR gate which is shown below.

A	B	C	IMPLY NOR [fJ]	MAGIC NOR [fJ]	MAJ(A, B, C)/MIN(A, B, C) [fJ]
0	0	0	2237.4341	133.861	0.00938889
0	0	1	2291.9421	44.9695	30.0444
0	1	0	2291.9421	44.9695	30.0439
0	1	1	2488.9182	45.0414	30.0427
1	0	0	2291.9421	44.9695	30.0439
1	0	1	2488.9182	45.0414	30.0444
1	1	0	2488.9182	45.0414	30.0351
1	1	1	2563.3306	45.0656	0.00267477

The average energy dissipations for the gate operation are 2392.9182 fJ for the IMPLY NOR gate, 56.1199125 fJ for the MAGIC NOR gate, and 22.53330796 fJ for the single Majority/Minority gate making the Majority/Minority logic gate (with ideal comparator) to be the most efficient logic primitive among the three mentioned above from the energy point of view.

4.3. Crossbar Mapping

The Majority MUGIC and Minority MUGIC logic gates can be mapped onto a memristor based crossbar array as shown in fig. 16 and fig. 17 respectively.

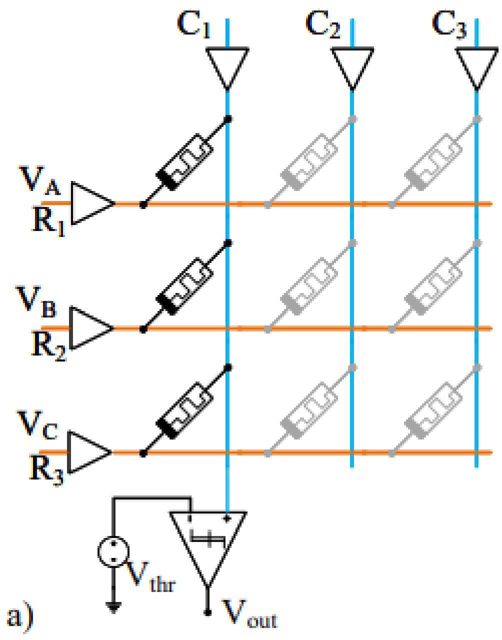


Fig. 16. Majority MUGIC logic gate mapped onto a crossbar array

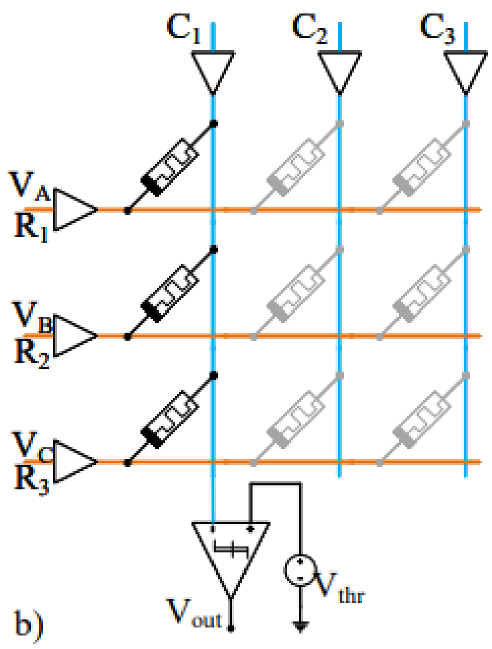


Fig. 17. Minority MUGIC logic gate mapped onto a crossbar array

CHAPTER 5.
MAPPING OF CIRCUITS ON CROSSBAR ARRAY

5.1. Proposed Pipeline

The proposed mapping pipeline is shown in fig. 18:

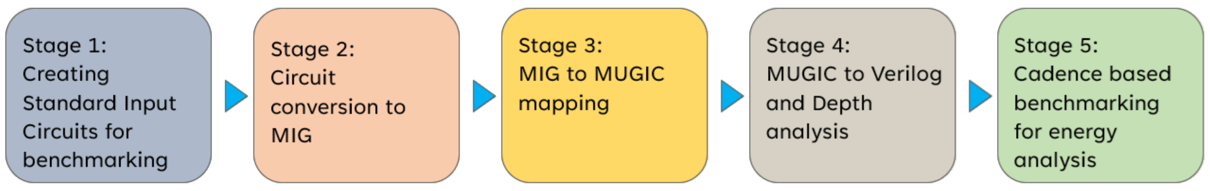

Stage 1: Creating Standard Input Circuits for benchmarking
Stage 2: Circuit conversion to MIG
Stage 3: MIG to MUGIC mapping
Stage 4: MUGIC to Verilog and Depth analysis
Stage 5: Cadence based benchmarking for energy analysis

Fig. 18. Proposed pipeline for crossbar circuit mapping

The standard input circuits used for benchmarking the algorithm are the ISCAS ’85 and ISCAS ’89 circuits. In Stage 1, we focused on implementing these circuits in the Berkeley Logic Interchange Format. Stage 2 uses the EPFL’s Mockturtle library to convert the benchmark circuit netlists into MIG (majority inverter graphs). Stage 3 goes a level deeper where device level memristors are mapped onto the crossbar array. Stage 4 and 5 are post processing steps where the benchmarks are measured for the metrics of delay, power, and area.

5.2. Algorithms

STAGE (2) ALGORITHM 1 : BLIF to MIG MAP

```
mig_network mig;
klut_network klut;
lorina::read_blif("Input_Files/blif/"+inp_file+".blif", blif_reader( klut )
); convert_klut_to_graph<mig_network>(mig,klut);
write_bench( mig, "Output_Files/MIG/"+inp_file[i]+".mig" );
```

STAGE (3) ALGORITHM 1: CROSSBAR MAP FROM MIG

```
void mapMIGToCrossbar(mig_network mig, Crossbar& crossbar) {
  // use a topologically sorted view of the MIG network
  topo_view mig_topo{mig};
  for (size_t i = 0; i < (mig.num_pis()*mig.num_gates()); i++) {
    for(size_t j = 0; j < mig.num_gates(); j++){
      crossbar.connectWires(i, j);
    }
  }

// Connect the i-th input to i-th row and j-th column of the crossbar
  crossbar.connectColumnsToFlipFlops();
// Iterate through the gates in the MIG network
//for (const auto& gate : mig.gates) {
// Mapping logic here
// Determine which crossbar columns to connect based on the gate's inputs
and outputs // Adjust the crossbar connections accordingly //
}
```

STAGE (3) ALGORITHM 2: CIRCUIT DEVICE DEFINITION

```
class memristor{
  private:
    double resistance;
    double voltage; // Applied voltage
    double memristance; // Value of internal Memristor resistance
    bool isLRS; // Resistance state
  public:
    memristor(): resistance(1.0),
voltage(0.0),memristance(0.1), isLRS(true) {}
  // Probing functions
    double getResistance() {
      return resistance;
    }
    bool isLRSState() {
      return isLRS;
    }
  // Configuration functions
    void switchState() {
      isLRS = !isLRS;
    }
    void setMemristance(double M) {
      memristance = M;
    }
    void applyVoltage(double V) {
      voltage = V;
      if (voltage > 0 && isLRS) {
        resistance -= memristance * voltage; // Adjust resistance in LRS
      }
      else if (voltage < 0 && !isLRS) {
        resistance += memristance * voltage; // Adjust resistance in HRS
      }
    }
    void disable(){
      isLRS = false;
    }
  };

class DFlipFlop {
  private:
    bool state;
  public:
    DFlipFlop() : state(false) {}
    void setInput(bool d) {
      state = d;
    }
    bool getOutput() {
      return state;
    }
 };
```

STAGE (3) ALGORITHM 3: CROSSBAR DEFINITION

```
class Crossbar {
  private:
    std::vector<std::vector<memristor>> crossbarArray;
    std::vector<DFlipFlop> outputflipflops;
  public:
      Crossbar(size_t numRows, size_t numCols) {
      crossbarArray.resize(numRows, std::vector<memristor>(numCols));
      outputflipflops.resize(numCols, DFlipFlop());
    }
     void connectWires(size_t row, size_t col) {
      crossbarArray[row][col].applyVoltage(1.0); // Apply a positive 
voltage to reduce resistance
    }
  // Disconnect two wires at a specific location in the crossbar
    void disconnectWires(size_t row, size_t col) {
      crossbarArray[row][col].applyVoltage(-1.0); // Apply a negative 
voltage to increase resistance
    }
  // Check if two wires are connected at a specific location
    bool areWiresConnected(size_t row, size_t col) {
      return crossbarArray[row][col].isLRSState();
    }
  // Set Memristance of a memristor at a specific location in the crossbar
    void setMemristance(size_t row, size_t col, double M) {
      crossbarArray[row][col].setMemristance(M);
    }
    void connectColumnsToFlipFlops() {
      if (crossbarArray[0].size() == outputflipflops.size()) {
        for (size_t col = 0; col < crossbarArray[0].size(); col++) {
          for (size_t row = 0; row < crossbarArray.size(); row++) {
            bool d 
= crossbarArray[row][col].isLRSState(); outputflipflops[col].setInput(d);
          }
        }
      }
      else {
        std::cout << "Number of flip-flops must match the number of 
crossbar columns." << std::endl;
      }
     }
     void printCrossbarState() {
       for (size_t i = 0; i < crossbarArray.size(); i++) {
         for (size_t j = 0; j < crossbarArray[i].size(); j++) {
           std::cout << (crossbarArray[i][j].isLRSState() ? '1' : '0') << " 
";
          }
         std::cout << std::endl;
       }
     }
     void printOutputs() {
       for (size_t i = 0; i < outputflipflops.size(); i++) {
         std::cout << "D Flip-Flop " << i << " Output: 
" << outputflipflops[i].getOutput() << std::endl;
       }
     }
   };
```

Stages 4 and 5 are currently under development.

CHAPTER 6.

Conclusion and Future Work

6.1. Conclusion

Based on the work above, there is a lot of room in this area to explore as this seems to be the future of computing. Memristor based emerging technologies like ReRAM, PCM RAM, and STT RAM seem quite promising. In this project, a new memristor based primitive was developed, compared against existing primitives, and a mapping pipeline with algorithms is developed.

6.2. Future Work

Stages 4 and 5 of the crossbar mapping pipeline is the scope of our future work, and finally, a simulator for spiking neural networks using a framework like Pytorch is currently in progress.

REFERENCES

[1] Electric field controlled semiconductor device, by Kahng Dawon, (1963, Dec. 27). Patent US3102230A.

[2] "TWINSCAN NXE:3600D," *ASML*. [Online]. Available: https://www.asml.com/en/products/euv-lithography-systems/twinscan-nxe-3600d. (Accessed: 29-Apr-2023).

[3] N. H. E. Weste, D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective,* 4th ed. Addison Wesley, 2011, pp. 55.

[4] G. E. Moore. Cramming more components onto integrated circuits, in *Electronics*, vol. 38, April, 1965.

[5] By, "Moore's law," *Intel*. [Online]. Available: https://www.intel.com/content/www/us/en/history/virtual-vault/articles/moores-law.html. (Accessed: 29-Apr-2023).

[6] M. Roser, H. Ritchie, and E. Mathieu, "What is Moore's law?," *Our World in Data*, 28-Mar-2023. [Online]. Available: https://ourworldindata.org/moores-law. (Accessed: 29-Apr-2023).

[7] I. Amato, C. Oldham, A. E. Lopez, R. Carpenter, *DARPA Defense Advanced Research Projects Agency 1958-2018,* Tampa, Florida, Faircount Media Group, 2018.

[8] R. Radisavljevic, A. Radenovic, J. Brivio, V. Giacometti, and A. Kis, "Single-layer MoS_2 transistors," in *Nature Nanotechnology*, vol. 6, pp. 147-150, Jan. 2011. doi: 10.1038/nnano.2010.279.

[9] S. J. Tans, A. R. M. Verschueren, C. Dekker, "Room-temperature transistor based on a single carbon nanotube," in *Nature,* vol. 393, pp. 49-52, May 1998. doi: 10.1038/29954.

[10] C. Mead, "Neuromorphic electronic systems," in *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629-1636, Oct. 1990, doi: 10.1109/5.58356.

[11] 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.

[12] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao, S. Joshi, H. Wu, H. S. P. Wong, and G. Cauwenberghs, "A compute-in-memory chip based on resistive random-access memory," in *Nature,* vol. 608, pp. 504-512, Aug. 2022, doi: 10.1038/s41586-022-04992-8.

[13] J. Reuben, "Rediscovering Majority Logic in the Post-CMOS Era: A Perspective from In-Memory Computing," *Journal of Low Power Electronics and Applications*, vol. 10, no. 3, p. 28, Sep. 2020.

[14] J. Reuben, "Binary Addition in Resistance Switching Memory Arraby Sensing Majority," *Micromachines*, vol. 11, no. 5, p. 496, May 2020.

[15] L. Cheng, M.Y. Zhang, Y. Li, Y.X. Zhou, Z.R. Wang, S.Y. Hu, S.B. Long, M. Liu, X.S. Miao, "Reprogrammable logic in memristive crossbar for in-memory computing," Journal of Physics D: Applied Physics, vol. 50, 2017.

[16] S. Kvatinsky, G. Satat, N. Wald, E. G. Friedman, A. Kolodny and U. C. Weiser, "Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 10, pp. 2054-2066, Oct. 2014.

[17] S. Ganjeheizadeh Rohani, N. Taherinejad and D. Radakovits, "A Semiparallel Full-Adder in IMPLY Logic," in *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 28, no. 1, pp. 297-301, Jan. 2020.

[18] A. Siemon, R. Drabinski, M.J. Schultis, X. Hu,; E. Linn, A. Heittmann, R. Waser, D. Querlioz, S. Menzel, J.S. Friedman, “Stateful Three-Input Logic with Memristive Switches,” Scientific Reports, vol. 9, 2019.

[19] R.B., Wald, N., Talati, N., Kvatinsky, S. SIMPLE MAGIC: Synthesis and In-memory Mapping of Logic Execution for Memristor-aided Logic. In Proceedings of the 36th International Conference on Computer-Aided Design, ICCAD ’17, Irvine, CA, USA, 12–16 November 2017; pp. 225–232.

[20] J. Reuben,; N. Talati,; N. Wald,; R. Ben-Hur,; A.H. Ali,; P.E. Gaillardon,; S. Kvatinsky, “A Taxonomy and Evaluation Framework for Memristive Logic,” In: L. Chua, G. Sirakoulis, A. Adamatzky, (eds) Handbook of Memristor Networks, Springer, Cham, Switzerland, pp. 1065–1099, 2019.

[21] P. Huang, J. Kang, Y. Zhao, S. Chen, R. Han, Z. Zhou, Z. Chen, W. Ma, M. Li, L. Liu, “Reconfigurable Nonvolatile Logic Operations in Resistance Switching Crossbar Array for Large-Scale Circuits,” Adv. Mater., vol. 28, pp. 9758–9764, 2016.

[22] J. Reuben and S. Pechmann, ”A Parallel-friendly Majority Gate to Accelerate In-memory Computation,” in Proc. IEEE 31st Int. Conf. on Application-specific Systems, Architectures and Processors (ASAP), Manchester, UK, 2020, pp. 93-100.

[23] L. Amarú, P. -E. Gaillardon and G. De Micheli, "Majority-Inverter Graph: A novel data-structure and algorithms for efficient logic optimization," *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, San Francisco, CA, USA, 2014, pp. 1-6.

[24] P. -E. Gaillardon *et al.*, "The Programmable Logic-in-Memory (PLiM) computer," *2016 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, 2016, pp. 427-432.

[25] S. Shirinzadeh, M. Soeken, P. -E. Gaillardon and R. Drechsler, "Logic Synthesis for RRAM-Based In-Memory Computing," in *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 37, no. 7, pp. 1422-1435, July 2018.

[26] D. Bhattacharjee, A. Easwaran and A. Chattopadhyay, "Area-constrained technology mapping for in-memory computing using ReRAM devices," *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, Chiba, Japan, 2017, pp. 69-74.

[27] V. Lakshmi, J. Reuben and V. Pudi, "A Novel In-Memory Wallace Tree Multiplier Architecture Using Majority Logic," in *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 3, pp. 1148-1158, March 2022.

[28] G. Jaberipur, B. Parhami and D. Abedi, "Adapting Computer Arithmetic Structures to Sustainable Supercomputing in Low-Power, Majority-Logic Nanotechnologies," in *IEEE Transactions on Sustainable Computing*, vol. 3, no. 4, pp. 262-273, 1 Oct.-Dec. 2018.

[29] V. Pudi, K. Sridharan and F. Lombardi, "Majority Logic Formulations for Parallel Adder Designs at Reduced Delay and Circuit Complexity," in *IEEE Transactions on Computers*, vol. 66, no. 10, pp. 1824-1830, 1 Oct. 2017.

APPENDIX

All the simulation files are on the EEE CAD server. The specific locations are listed as follows:

- eee-cad-server: /home/rahul_singh
- eee-cad-server: /home/2020AAPS0114U
- eee-cad-server: /home/2020A3PS0115U