Project:

MNEMOSENE

(Grant Agreement number 780215)

"Computation-in-memory architecture based on resistive devices"

Funding Scheme: Research and Innovation Action

Call: ICT-31-2017 "Development of new approaches to scale functional performance of information processing and storage substantially beyond the state-of-the-art technologies with a focus on ultra-low power and high performance"

Date of the latest version of ANNEX I: 17/09/2020

D4.5 – Refined memristor crossbar based logic and memory design and models

Project Coordinator (PC):	Prof. Said Hamdioui Technische Universiteit Delft - Department of Quantum and Computer Engineering (TUD) Tel.: (+31) 15 27 83643 Email: S.Hamdioui@tudelft.nl
Project website address:	www.mnemosene.eu
Lead Partner for Deliverable:	IMEC
Report Issue Date:	26/12//2020

Document History *(Revisions – Amendments)*	
Version and date	**Changes**
30-6-2020	First draft version
23-12-2020	Second and full draft version
26-12-2020	Final version reviewed by project coordinator

Dissemination Level		
PU	Public	**X**
PP	Restricted to other program participants (including the EC Services)	
RE	Restricted to a group specified by the consortium (including the EC Services)	
CO	Confidential, only for members of the consortium (including the EC)	

The MNEMOSENE project aims at demonstrating a new computation-in-memory (CIM) based on resistive devices together with its required programming flow and interface. To develop the new architecture, the following scientific and technical objectives will be targeted:

The MNEMOSENE project has received funding from the European Union's Horizon 2020 Research and Innovation Programme under grant

- Objective 1: Develop new algorithmic solutions for targeted applications for CIM architecture.
- Objective 2: Develop and design new mapping methods integrated in a framework for efficient compilation of the new algorithms into CIM macro-level operations; each of these is mapped to a group of CIM tiles.
- Objective 3: Develop a macro-architecture based on the integration of group of CIM tiles, including the overall scheduling of the macro-level operation, data accesses, inter-tile communication, the partitioning of the crossbar, etc.
- Objective 4: Develop and demonstrate the micro-architecture level of CIM tiles and their models, including primitive logic and arithmetic operators, the mapping of such operators on the crossbar, different circuit choices and the associated design trade-offs, etc.
- Objective 5: Design a simulator (based on calibrated models of memristor devices & building blocks) and FPGA emulator for the new architecture (CIM device combined with conventional CPU) in order demonstrate its superiority. Demonstrate the concept of CIM by performing measurements on fabricated crossbar mounted on a PCB board.

A demonstrator will be produced and tested to show that the storage and processing can be integrated in the same physical location to improve energy efficiency and also to show that the proposed accelerator is able to achieve the following measurable targets (as compared with a general purpose multi-core platform) for the considered applications:

- Improve the energy-delay product by factor of 100X to 1000X
- Improve the computational efficiency (#operations / total-energy) by factor of 10X to 100X
- Improve the performance density (# operations per area) by factor of 10X to 100X

Table of Contents

1. Introduction

Emerging computation-in-memory (CIM) architectures using memristor-based technologies are promising as they can enhance the computation efficiency, solve the data transfer bottleneck and at the same time deliver high energy efficiency using the normally-off/instant-on properties of their devices. In a CIM architecture, computational tasks are performed within the memory itself by exploiting arrays of devices to perform multiply-and-accumulate (MAC) operations on a physical level, which provide a computational platform for many outstanding applications such as pattern matching, voice recognition, classification, image processing, etc. Besides MAC operations, the CIM architecture also allows for logic computations, e.g. using the Scouting Logic principle.

The storage unit in a CIM architecture is a highly compact crossbar structure built using non-volatile, scalable, and CMOS-compatible memristor devices, such as Resistive Random-Access Memories (RRAM), Phase-Change Memory (PCM) or Magnetoresistive Random Access Memory (MRAM). Data in these memristive devices is stored as resistance states, which places the data access in the analog domain, whereas the surrounding data communication remains digital.

Hence, the overall computing efficiency of CIM systems depends not only on proficiency in analog computation but also on the performance of the conversions between analog and digital data streams. Where conventional architectures suffer from the memory to processor bottleneck, CIM using memristive devices has to deal with a conversion bottleneck. Analog/digital converters (ADC's and DAC'S) have been identified as an essential block in the CIM computing system that governs and thereby limits the speed, power/energy, and accuracy of the CIM operations. Further, on the CIM tile level, additional peripheral circuits and control logic has to be considered.

This deliverable has different sections dealing with these different CIM aspects:

Section 2 investigates, on the lowest hardware level, the impact of the memristor array architecture and ADC design choices on the performance of MAC operations for ReRAM based memsristor devices. As for the array architecture, standard 1T1R, vertical 1T1R and pseudo-crossbar 1T1R designs are compared, and the influence of array parasitics (R,C) investigated. 2 different ADC designs (delay-based and ring-oscillator based) are proposed.

Section 3 reports on the optimization of a CIM tile, based on an STT-RAM memristor array, for performing binary logic operations. Results are presented here on the level of the CIM core, consisting of STT-MRAM crossbar array and Periphery: control logic, row/column decoders and drivers, sense amplifiers (SA), registers, flip-flops etc.

Section 4 reports on the optimization of a CIM tile, based on an STT-RAM memristor array, for performing Matrix-Matrix Multiplication (MMM), or Vector Matrix Multiplication (VMM) operations. While Section 2 reports on the basic MAC operation, for MMM operations, besides the array and ADC, additional periphery is required as row-decoder DACs, post-processing circuits, and a control block

Finally, where both Section 2 and 4 use binary weights stored in binary memristors, **Section 5** describes an approach to store multiple bits of an integer number into the memristor array and perform the additional steps necessary for an integer MMM operation. The addition operations will be partially performed in the analog array and partially in the digital periphery.The proposed design utilizes minimum-sized adders and is customizable to support varying numbers of ADCs.

2. Simulation of performance metrics (Energy & Speed) for MAC operation in ReRAM based arrays (white box model)

2.1 Read-out structure

To perform a MAC operation with two operands (matrices), the first matrix is applied to the crossbar as voltage signals via Digital-to-Analog (DAC) interfaces, and the second matrix is stored in the crossbar as the resistance or conductance of the resistive memories. Based on Ohm's Law, the current passing through each resistive memory is the multiplication of the input voltage and the conductance of the resistive memory, and based on the Kirchhoff's Current Law, the current passing through the Bitline is the summation of these multiplications.

The current passing through the Bitline is an analog signal and contains the information of the multiplication and accumulation. To enter the digital domain, it needs to be converted by an ADC. This ADC is one of the major bottlenecks of this method for the acceleration of Matrix-Matrix-Multiplications (MMM) acceleration [1] which are build up from many MAC operations. There are two general ideas for the ADC:

1. *Using a fast, high resolution, high power consumption and large ADC*: Although such an ADC can produce the complete output for one column in each activation cycle, it must be shared between multiple bit-lines [2, 3, 4], this time-multiplexing scheme results in an increased latency. It is worth mentioning here, that the number of Bitlines, which an ADC shares also depends on the area and power constraints of the whole system. Also, by leveraging "analog buffers", the number of the ADCs can be decreased [1, 2].
2. *Using a slow, low resolution, low power consumption and small ADC:* With such an ADC, having one ADC per bit-line is possible, [5] and [6] are using this approach consisting of a "Sense Amplifier" and "Integrate and Fire" circuit. Both interfaces can produce 1 bit per activation cycle, i.e. for generating n-bit output, they need 2^n comparison cycles, which still results in a long latency.

At TU Delft, we developed two compact and power-efficient ADC designs that have the possibility of being integrated per Bitline.

In this work, 2 different ADC's designs are proposed and studied

2.1.1 Delay-based ADC

In the initial phase, we propose a delay-based ADC. **Figure 1**. shows the schematic of the ADC.

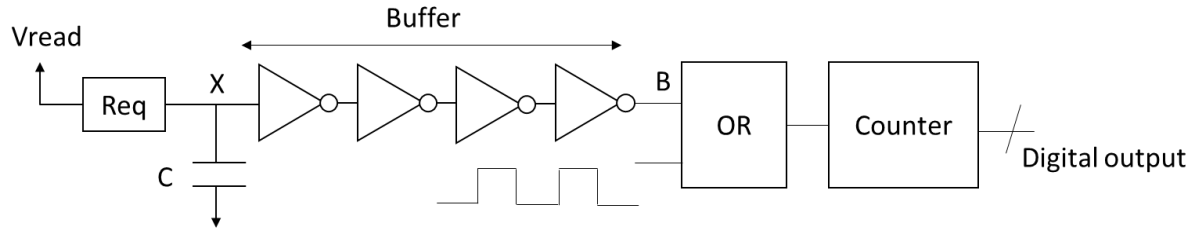

Figure 1. The schematic of the proposed delay-based ADC.

The ADC must differentiate different equivalent resistances (R_{eq}) resulting from different combinations of Low Resistive States (LRS) and High Resistive States (HRS) which are selected in a column of the 1T1R crossbar. The underlining idea is the time it takes to charge the capacitor *C* depends on R_{eq} (it is equal to $R_{eq} * C$). By supplying a row voltage (V_{read}) to the crossbar, a column current starts to charge *C* from 0 V to V_{read}. The voltage over *C* is passed through a buffer (four NOT gates in this case) to get a square waveform. The duration that the signal takes to arrive at node "B"

as logic “0” represents the delay which is a function of the resistances in the crossbar and encodes an element of the result matrix. This delay can be digitized by ORing it with a clock pulse, the number of pulses can then be counted by a counter and the output of the counter will be the digital value of the corresponding element of the result matrix. For an initial investigation of the circuit performance, it was simulated using the parameters in **Table I**. The delay of the signal at node B for the resistance combination {3*LRS & 0*HRS } was 116 ps and for the resistances {2*LRS & 1*HRS} was 169ps. To distinguish these two states from each other, a clck pulse with a period of at least 53 ps (>18 GHz) needs to be ORed with the output of the buffer. If more devices are read out at the same time the required frequency of the clock increases. Having such an ultra-high frequency on the chip makes counting pulses challenging.

Table I: Simulation parameters for the first investigation of the ADC.

Parameters	Specifications
RRAM devices	HfO_x
R_{off}/R_{on}	30 kΩ / 3 kΩ = 10
Simulator	Synopsys HSPICE
CMOS technology	TSMC 90 nm
Read Voltage	1.3 V
CMOS Specs	TT 27C
Counter Voltage	1V
C	180 fF

2.1.2 Ring-oscillator based ADC

To convert the analog output signal of the crossbar to a digital signal, we decided to use a VCO-based ADC which is a time-based ADC, therefore, it can provide the advantages of the time-based signals with a relatively easy design procedure. To transfer an analog current into the digital signal with the help of VCO-based ADC, three stages are required: at the first stage, the analog bit-line current needs to be transformed into an analog voltage, after this stage, obtained analog voltage is transformed into pulses with the help of the VCO and finally generated pulse are counted with a counter and mapped to the corresponding digital signal, output of this stage is the equivalent digital signal and can be processed by the digital host. The schematic of the whole system including crossbar that produces an analog current and the VCO-based ADC is shown in **Figure 2**.

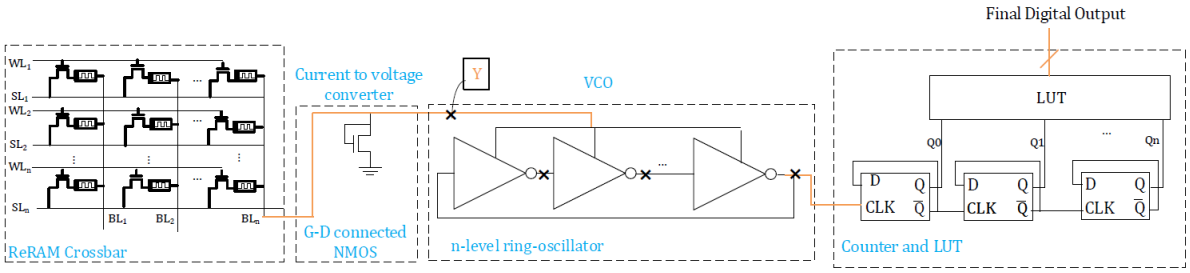

Figure 2. Detailed schematic of VCO-based ADC

The functionality of the ADC is as follows: the resistive crossbar modulates the ring oscillators frequency. This frequency modulation is observed in the changed number of pulses that are generated. The pulse period is defined as the 50 % rise fall time at the node VCO outpout.

Different resistance configurations (LRS/HRS) in the array result in different VCO supply voltages (V_y) and so a different oscillation frequency= 1/(pulse period)

In **Figure 3**, voltage value at node Y (V_Y) and number of pulses generated by the ring-oscillator are presented with increase in number of LRS devices in a column. Different “current-to-voltage converters” are shown i.e. 1) No converter used 2) G-D shorten NMOS having diode configuration 3) fixed resistors of value 5k and 4) 10k Ohms. Diode-based current-to-voltage converter provides better input-output characteristcs as it has the most dynamic range. Therefore, diode-based converter is used for further analysis.

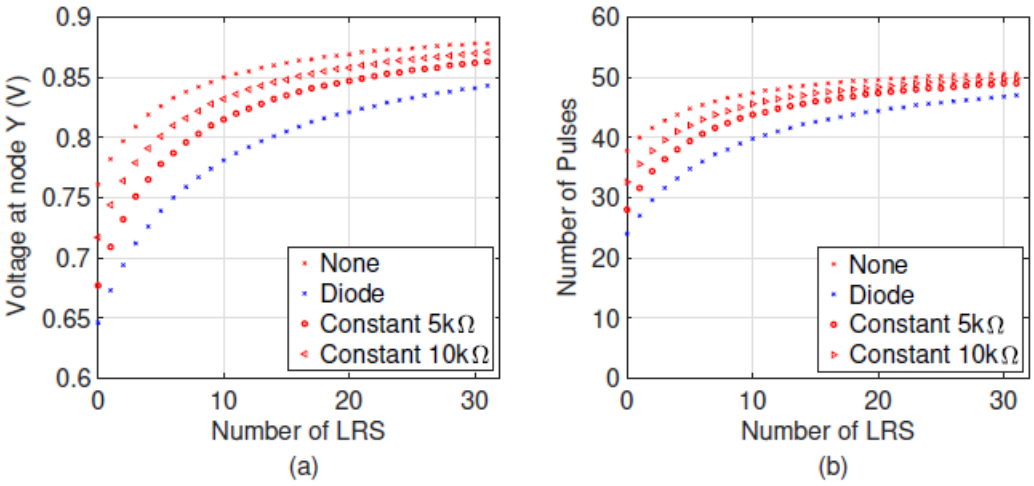

Figure 3. (a) Voltage developed at node Y in Fig.2 and (b) Resolution of the VCO-based ADC with different method ("None": no extra device, “Diode”: diode-connected structure and “5k”, “10k”: two different values of constant resistances).

2.2 Array simulations

The CIM tile consists of the Memristive Crossbar as well as analog and digital peripheral circuits. Here, our focus will be on the memristive crossbar as well as on the readout circuitry. The ADC’s described in the previous chapter are used as a readout circuit to perform Vector Matrix Multiplication in a realistic 1T1R crossbar structure. The considered array structures were 1T1R array, Vertical 1T1R array, and Pseudo-Crossbar array. One column per array structure is shown in **Figure 4**. On our level of investigation, the difference between these structures lies mainly in the way how they are utilized to do computation. For convention, the metal lines connecting the transistor gates are denoted as Word lines (WL), the lines connecting the Crossbar to the readout circuitry are called Bit lines (BL) and the lines connecting to the remaining transistor contacts are called Source lines (SL). The general way to perform computation has been described in **Deliverable 4.4**. Therefore, it will just be briefly repeated. In the 1T1R array, the common Source line is set to the read voltage (ADC1 0.7 V; ADC2 1 V), while the input signals are applied to the Word lines 1 to 8 either as logical ‘1‘ (ADC1 1.3 V; ADC2 1.5 V) or ‘0‘ (0 V in both cases). In the vertical 1T1R array, the common Word line is set to 1 (1.3 V or 1.5 V) while the input signals are applied to the Source lines 1 to 8 either as logical ‘1‘ (0.7 V or 1 V) or as logical ‘0‘ (high ohmic). In the pseudo crossbar, both modes of operation are possible.

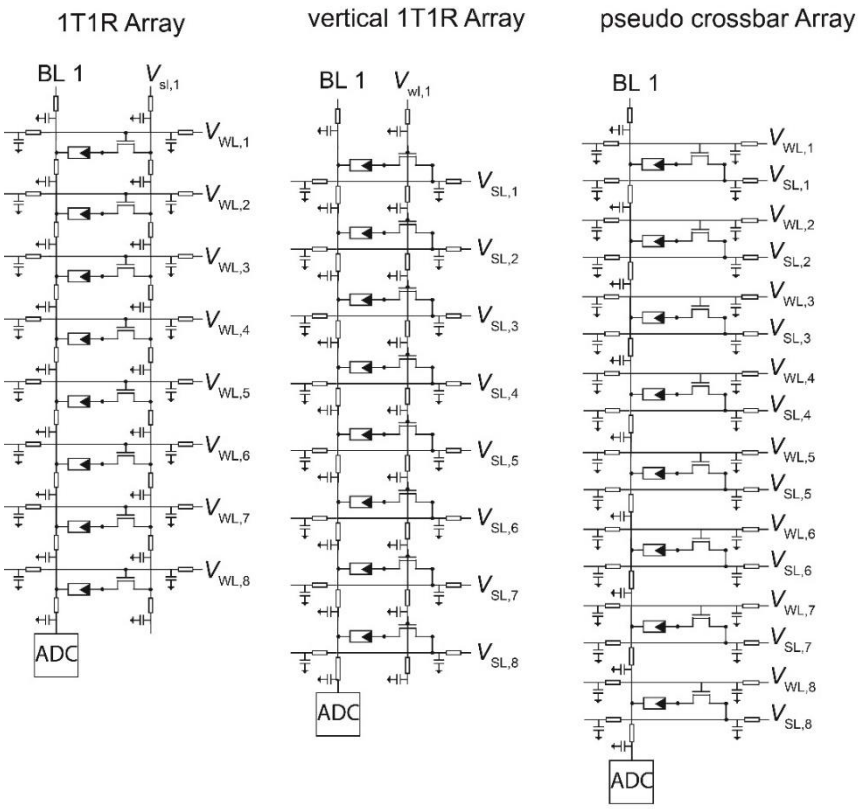

Figure 4: Schematics of one column of the different memristive crossbar configurations showing the connection to the readout circuitry.

2.2.1 Array simulations using Delay-based ADC output structure

Simulation parameters

For the array level simulations, we used the JART VCM v1b deterministic model, which was calibrated with 100 nm x 100 nm HfO_x devices. The LRS was chosen as 50 kΩ and the HRS as 2.5 MΩ. The ADC1 from the previous chapter was redesigned in a 40 nm CMOS node by TSMC. It can detect 9 levels (8 rows + zero). The transistors in the crossbar were chosen as low V_{th} with dimension $L = L_{min} = W/10$. The WLs, SLs, and BLs have parasitic resistances of 1 Ω/segment and capacities of 210 aF. Those values are based on FEM simulations of a 45 nm DRAM crossbar structure.

Simulation Results

Through the simulations, different results could be generated. Some results concern the properties of the devices and the crossbar organization while others concern the ADC. First, the device/ crossbar level results will be shown and at the end, the ADC results will be shown. The effective resolution of the ADC denotes the number of devices that can be read out in parallel at the same time with one ADC. From **Figure 5** it can be seen that it is a function of the R_{off}/R_{on} ratio. The simulations were performed for a fixed R_{on} of 50 kΩ while the R_{off} was varied to realize the different resistance ratios. It can be observed that increasing the R_{off}/R_{on} ratio generally improves the ADC resolution. However, increasing it over a certain limit (R_{off} = 2 MΩ) leads to no additional benefit.

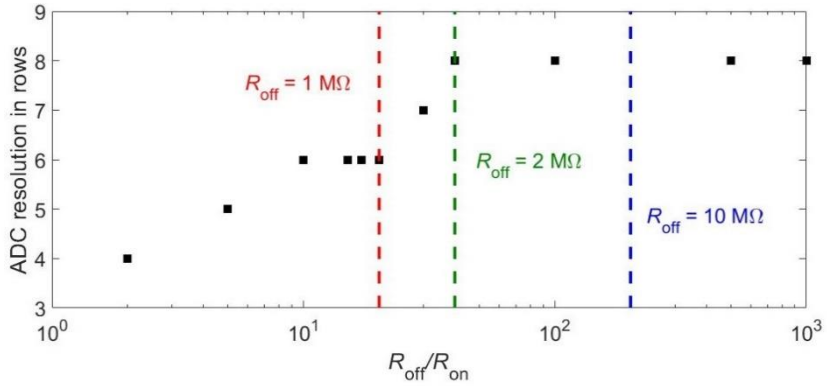

Figure 5: The ADC resolution as a function of the R_{off}/R_{on} ratio.

The purpose of the ADC is to differentiate the stored data in the array and to provide a unique result for different data. For an ADC that can differentiate a maximum of 8 rows at a time, this means that it has to provide different outputs for 8 HRS/0 LRS, 7 HRS/ 1 LRS up to 0 HRS/ 8 LRS devices. These cases represent the different resistance configurations of the part of the crossbar that is read out. They can be characterized as having different equivalent resistances where the equivalent resistance is determined by the parallel connection of the memristive devices in the LRS or HRS state. **Figure 6** shows the R_{eq} of the crossbar for all resistance configurations using 8 devices and for different R_{off}/R_{on} ratios. It can be observed that a higher R_{off}/R_{on} ratio leads to a stronger nonlinearity approaching a 1/#LRS behavior while a lower R_{off}/R_{on} ratio leads to a linearization of the R_{eq} characteristic. For larger numbers of LRS devices, the R_{eq} are more or less independent of the R_{off}/R_{on} ratio since they only depend on the LRS values. The difference between neighbouring resistance configurations decreases for an increase in the number of LRS devices and it becomes smaller for smaller R_{off}/R_{on} ratios. This points towards a problem when using a conventional ADC since they usually have a linear input to output relationship.

The results of this crossbar ADC simulation concerning latency and energy are now as follows. For a constant cycle time, one evaluation of 8 devices in one column takes 40 ns to perform. The energy can be split up into three components, namely the energy for the WL drivers, the energy for the SL drivers, and the energy consumption of the ADC itself. Between those parts, the WL drivers show negligible energy consumption in the range of a few attojoules, the SL drivers show an energy consumption of around 80 fJ while the ADC shows the highest energy consumption of up to 700 fJ if all the devices are in the LRS. The total data-dependent energy consumption and latency are shown in **Figure 7**. It can be seen that it is data-dependent with higher energy consumption for a higher number of LRS devices that are being readout.

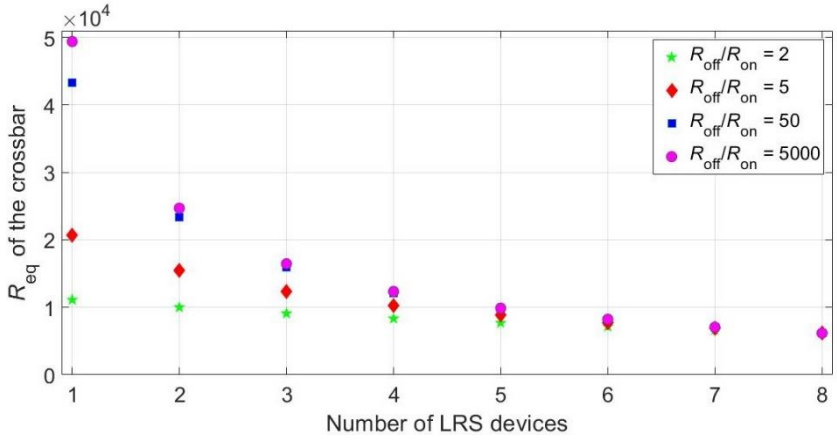

Figure 6: Equivalent resistance of the crossbar for different numbers of devices in the LRS.

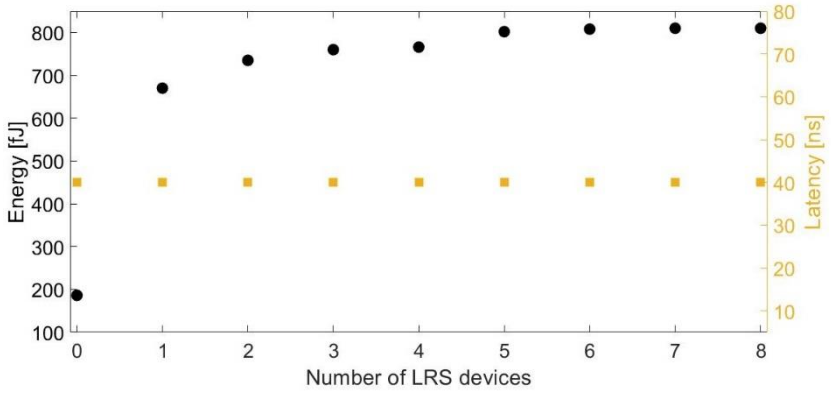

Figure 7: Energy and latency numbers of the proposed ADC and crossbar for 8 Vector Matrix Multiplications.

An alternative approach to operate the ADC is to utilize the data-dependent latency since the ADC will finish its computation faster for a higher number of LRS devices. The results for this operation mode can be seen in **Figure 8**. This operation mode reduces the average latency from 40 ns to 16 ns and also leads to a strong reduction of the used energy. One interesting feature of this operation mode is that for a larger number of LRS devices, the energy consumption can actually be reduced even though the crossbar becomes less resistive. The resulting higher current should lead to a higher power consumption. The reason for this is that the reduction in evaluation time of the ADC for more LRS devices overcompensates the increase of energy due to the lower ohmic crossbar.

Discussion

From our simulation results we can conclude that the ADC plays the most critical role for the system performance concerning energy and latency, while the crossbar plays a less critical role. Since the energy consumption depends on the data and thereby on the LRS and the HRS values, we can conclude that higher LRS as well as HRS states will be favorable from an energy point of view.

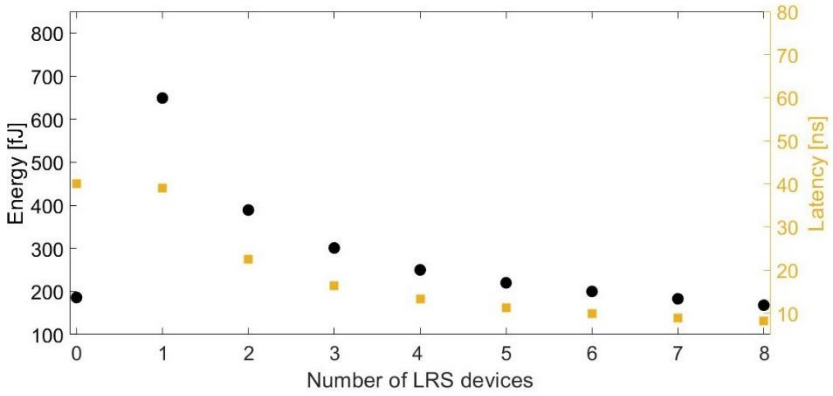

Figure 8: Energy and latency numbers of the proposed ADC and crossbar for 8 Vector Matrix Multiplications when using the data dependency of the ADC latency.

2.2.2 Array simulations using Ring-Oscillator based ADC output structure

Simulation Parameters

From the results on the VCO-type ADC (**Figure 9**) it can be seen that the levels/(time*energy) is better if a smaller resolution is chosen:

@ 12 levels~ 12 levels/(2ns*1pJ) =6 levels/ns*pJ

@ 32 levels ~ 32 levels/(10ns*5pJ)=0.64 levels/ns*pJ

Since we dont want 2.5 bit ADC, we choose the 3 bit = 8 quantization levels

Note that it is possible to count for a longer time to increase the difference between the levels: if 7 LRS produce 40 pulses in 1 ns and 8 LRS produce 41 pulses in 1 ns the difference is 1 but if we count for 2 ns the difference becomes 2.

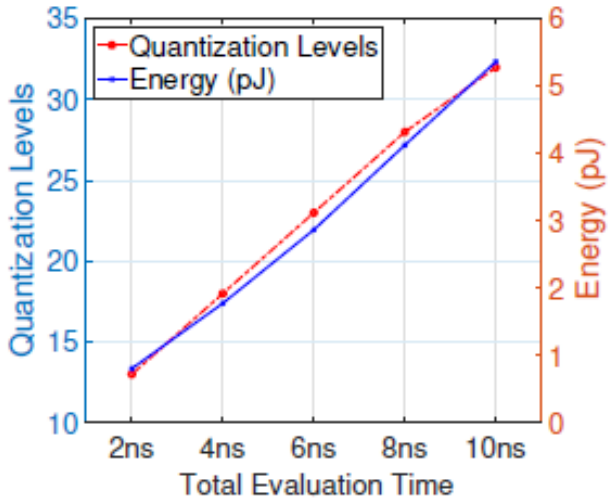

Figure 9 : Energy and Quantization levels vs. Evaluation Time for VCO type ADC.

As the VCO based ADC is designed using 28nm TSMC technology (while we used 40nm TSCM in 2.2.1), array simulations had to be adapted as well. As a 28nm TSMC technology PDK was not available at RWTH, we use a predictive 32nm technology for the ADC and the crossbar instead.

Since we used a different CMOS technology, we needed to fit the ADC parameters to distinguish 8 levels, see **Table III**.

The crossbar and device parameters used are listed in **Table IV**.

Table III ADC parameters

ADC parameter	Value
# of inverters in the ring oscillator	3
w_{pinv}	700 nm
l_{pinv}	350 nm
Diode NMOS size	$w = l$ =32 nm
V_{SL}	1 V
V_{WL}	1.5 V

Table IV: Crossbar and device parameters

Crossbar and device parameter	Value
LRS	1.6 kΩ @ 0.1 V
HRS	340 kΩ @ 0.1 V
w_{1T1R}	250 nm
l_{1T1R}	32 nm
$C_{Gate}=\varepsilon_o$*EPSROX/TOXE	30 aF*W/W_{min} *L_{min}/L = 240 aF
$R_{parasitic/cell}$	10 Ω
$C_{parasitic/cell}$	1 fF

As the ADC is used as a 3 bit ADC and connected to a column of the crossbar, it makes sense to consider crossbars with 8 rows and a large number of columns compared to an equal number of rows and columns.This is also more reasonable for applications like databases where there are more entries than characteristics per entry

One important question is now how large the array can be if we use a single driver per input. This depends among others on the driving requirements for the input drivers.

Design considerations and simulation results 1T1R array

During the readout the column Sourceline (SL) is activated and the inputs are applied to the different Wordlines (WL). This means we need one driver per Wordline

Increasing the number of columns increases the load that has to be driven by the WL drivers: per additional column we have one transistor gate per driver. Also the delay will vary in different columns, so this is a challenge for our parallel ADC approach.The SL drivers have a limited load, i.e. 8 1T1R cells in parallel. So the driver load varies column to column through the wordlines.

Figure 10(b) shows the number of pulses/ns as function of the number of LRS devices in the column. The number of pulses increases with the number of LRS but the distance between the pulses decreases.

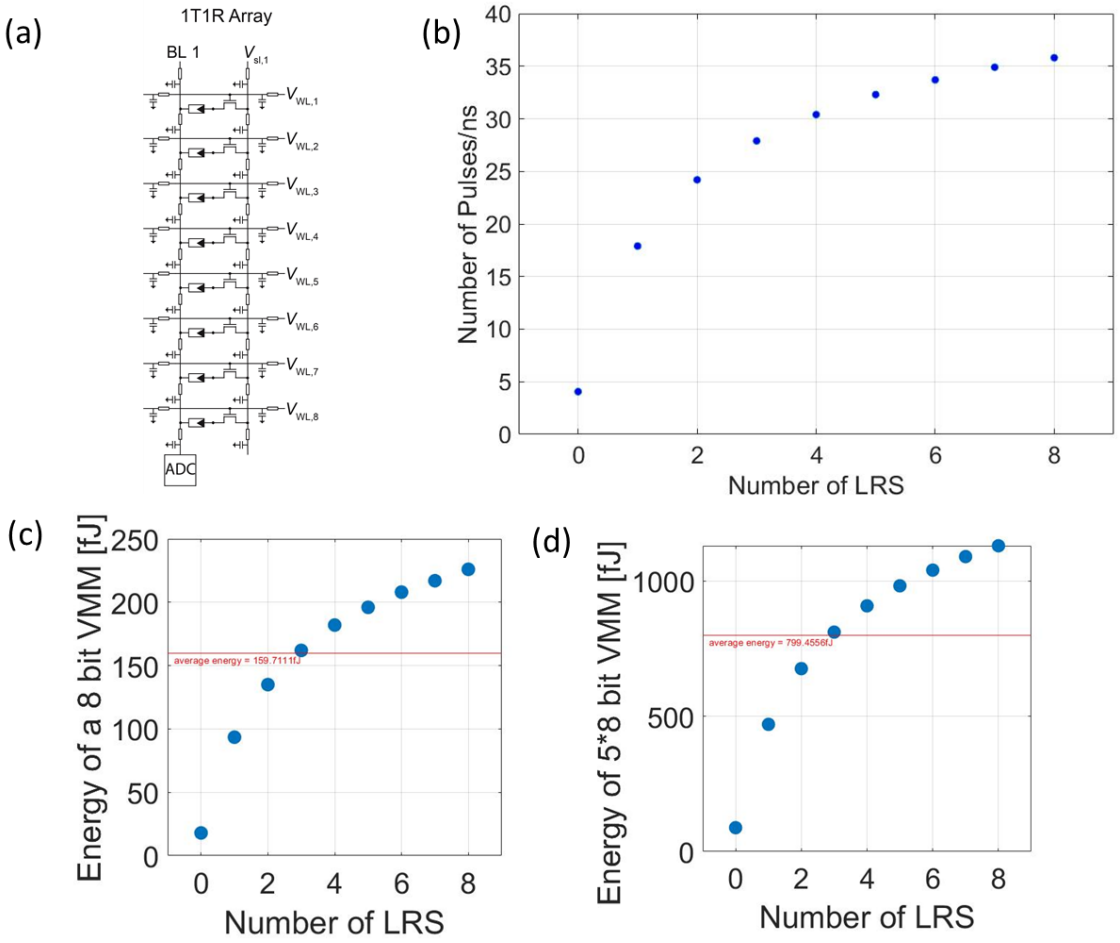

Figure 10: Results for 1T1R array: (a) 1T1R architecture, (b) number of ADC pulses (per ns) a.f.o. number of LRS states in a column, (c) Energy for reading out 1 column in parallel, (d) Energy for reading out 5 columns in parallel. The red horizontal lines in (c) and (d) depict the average energy.

Figure 10(c) shows the energy (= I*V*t = I*1 V*1 ns) for reading out 1 column in parallel. For the initial ADC1 design we had energy values between 200 fJ for 0 LRS (new ADC2: factor 10 improvement) and 800 fJ for 8 LRS, so the new ADC2 shows an improvement of a factor 10 resp. 3. Also here, ADC energy is dominant (energy for WL charging < 2 % of ADC energy). It should be mentioned here, that the LRS is much smaller for the second ADC therefore further improvement could be made by redesigning the ADC to operate on higher LRS values.

In order to evaluate how that energy scales for parallel readout, we examined the case of a crossbar with 41 columns and 5 readout circuits, i.e. 1 ADC per ~8 columns. As can be deduced from Fig. 8(d), the total energy scales linearly with the number of parallel reads.

Design considerations and simulation results Vertical 1T1R array

During the readout, the column Wordline (WL) is activated and the inputs are applied to the different Sourcelines (SL), this means we need one driver per Sourceline.

Increasing the number of columns increases the load that has to be driven by the SL drivers: per additional column one transistor drain per driver. The SL drivers have a constant load of 8 1T1R cells in parallel.

As shown in **Figure 11**, we get very similar results for the 41 column array as for the 1T1R architecture, i.e. energy consumption is the same in these cases.

Note that depending on the size of the parasitic elements, we need to consider a different setup time for the ring oscillator. During this time period the frequency is changing. Only after some time it reaches a constant frequency. For 41 columns, C_{par}=1fF and R_{par}=10 Ohm, this setup time is around ~500 ps.

Another way to handle the different frequency at the beginning would be to count for a longer time. Since the initial deviation would have a smaller influence then.

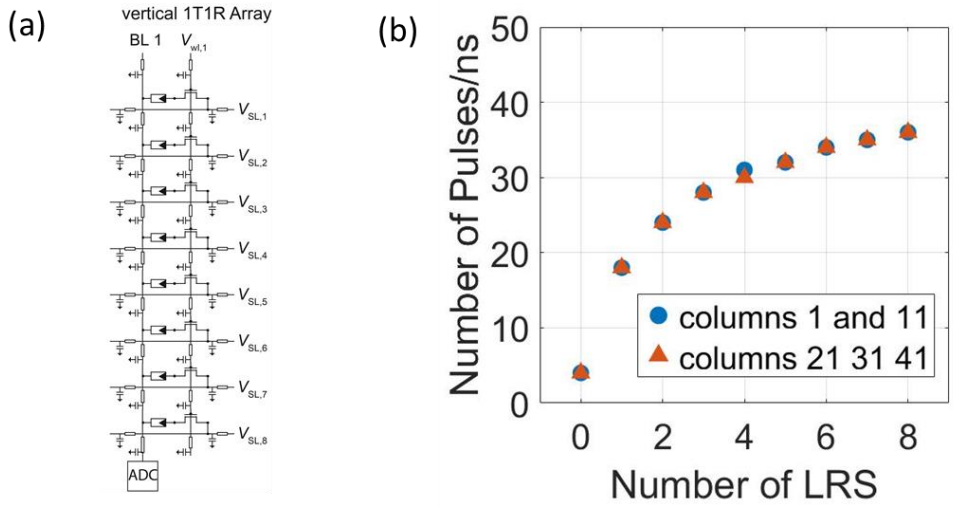

Figure 11: Results for Vertical 1T1R array: (a) Vertical 1T1R architecture, (b) number of ADC pulses (per ns) a.f.o. number of LRS states in a column, for different column numbers.

Design considerations and simulation results Pseudo Crossbar Array

During the readout the inputs can be applied via either the Word- or Sourcelines while the other is just set to a high voltage. Therefore, we can use the pseudo crossbar in different ways:

1) like the 1T1R array (inputs applied to WL and SL just active)

2) like a vertical1T1R array (inputs applied to the SL and WL activated)

3) apply the inputs to WL and SL

A comparison between variants 1, 2 and 3 was done using a simulationof an array with 26 columns (with readout at columns 1, 11, 21, 26), and using array parasitics of C_{par} = 10 fF, R_{par} = 10 Ohm. The results are shown in **Figure 12**.

We see a larger variation for the different columns for applying inputs to SL (**Figure 12 (c)**) and even worse for applying inputs to both WL and SL (**Figure 12 (d)**). Applying the inputs via the WL (**Figure12(b)**) gives the best results for the pseudo crossbar array case

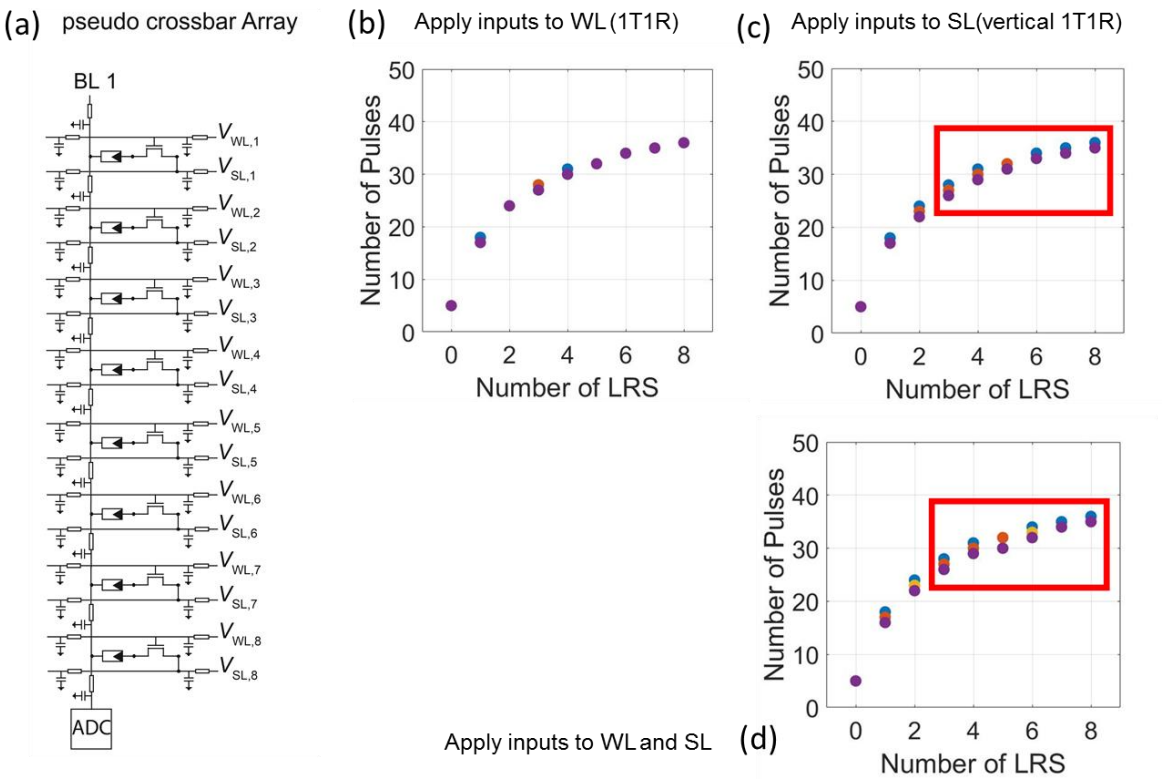

Fig.12: Results for Pseudo-Crossbar 1T1R array. (a) Pseudo Crossbar 1T1R architecture, (b-d): number of ADC pulses (per ns) a.f.o. number of LRS states in a column, for different column numbers and for applying inputs (b) to WL (similar to 1T1R array), (c) to SL (similar to vertical 1T1R array), and (d) to both WL and SL:

General results

From the comparison of the arrays we can see that it makes the crossbar more tolerant towards parasitics if we apply the inputs via the WL (transistor gates).The capacitances can be compensated if we handle the setup time.

Compared with the initial ADC design we are better by a factor of 3 to 10 for the energy values even though the LRS was considerably lower. Another big advantage is that the improved design offers the possibility to work on larger array sizes implying a larger parasitics tolerance through the use of the setup time.

Again most of the energy is consumed by the ADC. The energy for one 8 bit VMM = eight one bit operations. Energy/(bit op) is between 2 fJ and 30 fJ.

3. Optimized CIM-tile with STT-MRAM macro for binary logic

In this Section, CIM-based binary logic design using scouting logic or read assist technique is presented. The main idea is to execute logic operations using STT-MRAM-based memristor device technology and investigate its potential applications. The primitive logic operations performed are OR and AND, and operations such as NAND, NOR, NOT etc. can be performed by simply inverting these primitive logic designs.

Scouting logic design falls under CIM-P hybrid class, based on the classifications described in [1]. This implies that the discussed logic operations are performed in the crossbar array, but the result is generated in the periphery (P) and it requires some level of modification in the crossbar array (hybrid). Using scouting logic technique, a logic operation can be performed by reading/accessing two rows at the same time.

3.1 Preliminaries

CIM Core

Figure 13 shows the STT-MRAM-based CIM core. The core consists of 1) STT-MRAM crossbar array: storage and processing unit with 1T1R bit-cell, and 2) Periphery: control logic, row/column decoders and drivers, sense amplifiers (SA), registers, flip-flops etc.

The crossbar consists of 3-terminal 1T1R bit-cell, with terminals word line (WL), bit-line (BL) and select line (SL). Enabling WL selects a particular device for reading, writing or computing, implying that NMOS devices (1T have gate voltage WL) acts as a selector device. Enabling BL performs the required function on the selected device, which is connected to read/write drivers during their respective operating cycles. SL is generally grounded for write operations and connected to SA/ADC for read/compute operations. STT-MRAM device (1R) is a MgO-based spintronic device that is capable of storing 1-bit information, with 0 and 1 represented as low (anti-parallel magnetization) and high (parallel magnetization) conductance states.

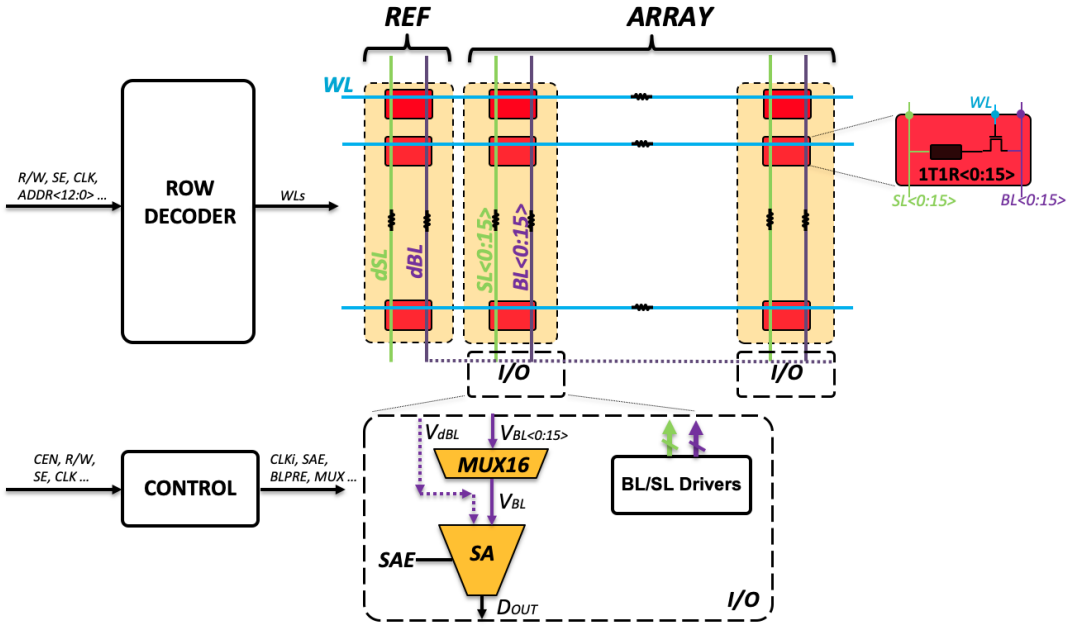

Figure 13: CIM Core: 1) A 32x512M16 STT-MRAM-based crossbar 2) Periphery: Control logic, row decoder, SA, BL/SL drivers.

The crossbar size (ARRAY in **Figure 13**) used is 32x512M16, and an Input/Output (I/O) unit is shared with 16 columns, implying 32 I/Os in total. A dummy column (REF in Figure 13) is used to produce reference voltage for read/compute operations, and the corresponding circuit and reference voltage values generated are described in **Figure 14**. Some of the important signals include CLK (external CLK), ADDR (external address), R/W to determine read or write operation, SAE (SA enable), BLPRE (bit-line pre-charge before each operation), MUX (to select one of the 16 columns for operation).

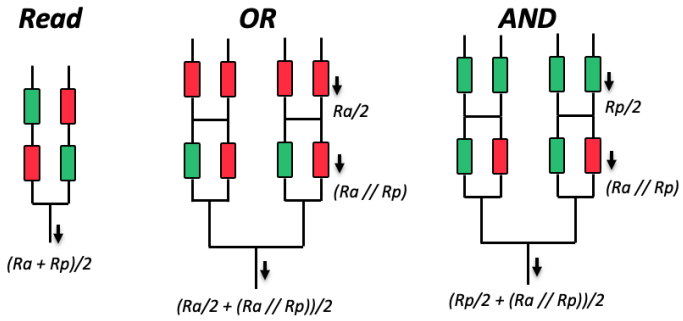

Figure 24: Reference circuit to generate reference voltages for read/logic operations.

The working of the SA is similar to that of SRAM-based SA. However, instead of sensing the difference in BL and NBL voltage of a 6T SRAM bit-cell to determine the state of the bit-cell, we use BL and dBL as two inputs of the SA. Here, BL voltage is developed by getting discharged through ON or OFF state of the STT-MRAM device, and the reference voltage is developed by the enabling the circuit (as shown in Figure 14) to generate a voltage that can differentiate the two states. **Figure 14** shows the reference generator circuit that differentiates the ON (1) and the OFF (0) state of the STT-MRAM device during read, differentiates OFF|OFF and OFF|ON during OR, and differentiates OFF|ON and ON|ON during AND. Below **Figure 15** shows the generated reference voltages along with BL voltages developed when the selected device is in State 0 or 1. Noteworthily, the SA to perform scouting logic operations needs modification as compared to a SRAM-based SA. This is because the voltage difference in BL/dBL corresponding to OR/AND operations are more stringent as compared to read operation in SRAMs (also seen in **Figure 15b**, the difference with respect to reference voltage is smaller in logic operations as compared to read operation).

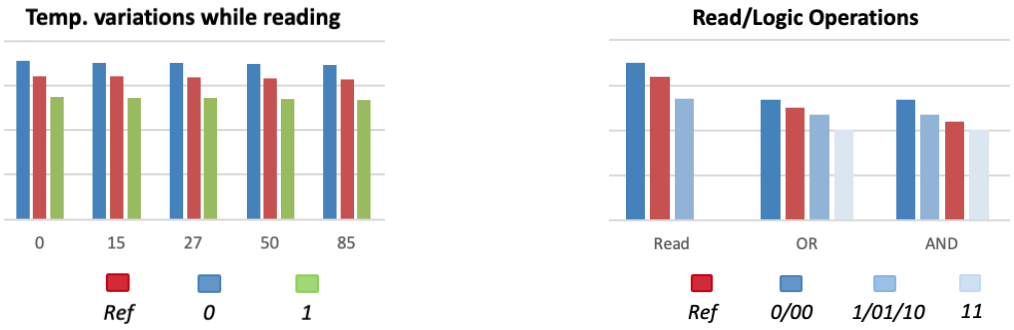

Figure 35: a) Normalized reference voltages with temperature variations during read operation. The SA is robust against temperature range of 0-85 degrees. b) Normalized reference voltages while reading and logic operations. The reference generated clearly differentiates the ON and OFF states of the STT-MRAM device.

Simulation Setup

Our intention is to perform post-layout simulations, using industry-standard 28nm CMOS device technology and accurate STT-MRAM experimental model. CIM core structure is based on fabricated chip, implying that all circuit components are experimentally verified. **Table V** describes parametric

values used for the simulations. Noteworthily, a high value of 0.9V is used as BL voltage to perform CIM-based logic operations. Using such a high voltage is power-inefficient and STT-MRAM device can suffer from read disturb. Based on our simulations, we can go up to 0.3V and still perform logic operations that can save significant crossbar power. It is, however, more prone to noise and reduces the dynamic range of currents used by the SA. As mentioned above, the SA spec is already more stringent than for a standard memory access, so this is not enabled yet with our current SA circuit design. As future work, a more robust design of SA is required to work with 0.3V and we will not yet report the data for this case in **Table V and VI**.

Table V: a) Parameters used for a) CMOS-based periphery and b) STT-MRAM devices to run the database query application related to health care.

Parameters	Values
MgO-based STT-MRAM	
Technology	Commercial 28nm
Periphery Voltage	0.9V
BL/SL Voltage	0.9V/0V
WL Read/Write Voltage	0.75V/1.1V
SA	Voltage-based

Parameters	Values
MgO-based STT-MRAM	
OFF/ON State	157K/90K
Resistance ratio	1.75
Crossbar Array	32x512M16
Column/SA	16
Bit-cell config	1T1R

3.2 Results

Targeted Application

Potential application that uses logic operations is database query, described in **Figure 16**.

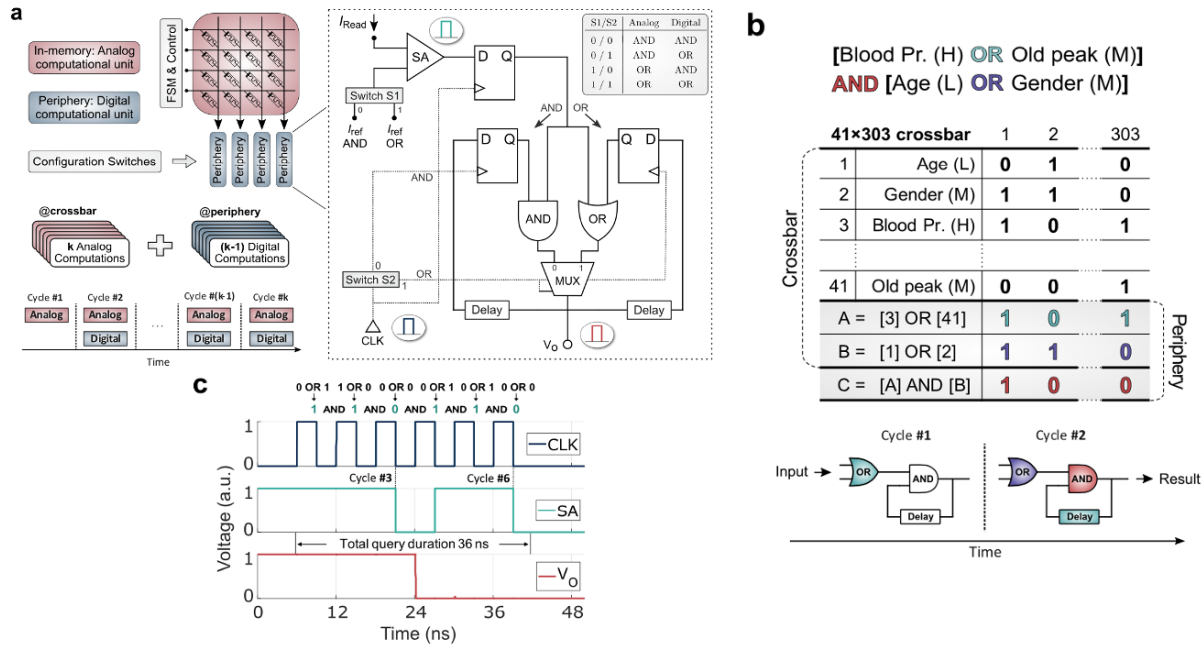

Figure 46: With reference to joint publication from TUD and IBM [34], a) Describes the cascaded logic design that can compute a series of logic operations b) The database query and its design implementation c) Waveforms related to the database query using PCM devices.

We investigated the promise of CIM-based logic designs while running a health care application that computes a large database to find potential patients, given their sex, age, cholesterol level, heart rate and other health parameters. Such tasks require a series of various bit-wise logic operations to solve a single query, and to determine whether a person is say, a target heart patient or not. The key challenge here to accumulate the intermediate results of logic operations into the final result. **Figure 16** shows the circuit implementation of the cascaded logic design that perform a, series of logic operations.

Simulation Results

Simulation results are summarized below in **Table VI**. Promising results are obtained using STT-MRAM device technology, with **37.9 TOPS/W** and **16.7 GOPS**, where one operation corresponds to one logical operation defined in the query. In other words, **26 fJ** per bit-wise logical operation.

Table VI: Results for STT-MRAM to run the database query application related to health care.

Params (STT-MRAM)	1 CIM Operation (1 Logic Op)	Database Query (11 Logic Ops)
Columns/SA	16	
Array Size	32x512	
Cycles Needed	16	96
Delay/cycle (ns)	3.5	21
Total Delay (ns)	**56**	**336**
Power Periphery (uW)	308	
Power Cascade (uW)	8.8	
Power Crossbar (uW)	51	
Total Power (uW)	**368**	
Energy Periphery (pJ)	17.25	103.5
Energy Cascade (pJ)	0.49	2.9
Energy Crossbar (pJ)	2.9	17.1
Energy Total (pJ)	20.7	123.6
Total Energy (pJ) (+20%)	**28.8**	**148.5**
Energy/Logic Op	**148.5pJ/512*11 = 26.4 fJ**	
GOPS	**16.7**	
TOPS/W	**37.9**	

Figure 17 shows the power shares of periphery circuit (SA, row drivers, row/column decoders, MUX, CLK generators etc.), STT-MRAM-based memristor crossbar array and the cascaded logic design to compute the heath care database query.

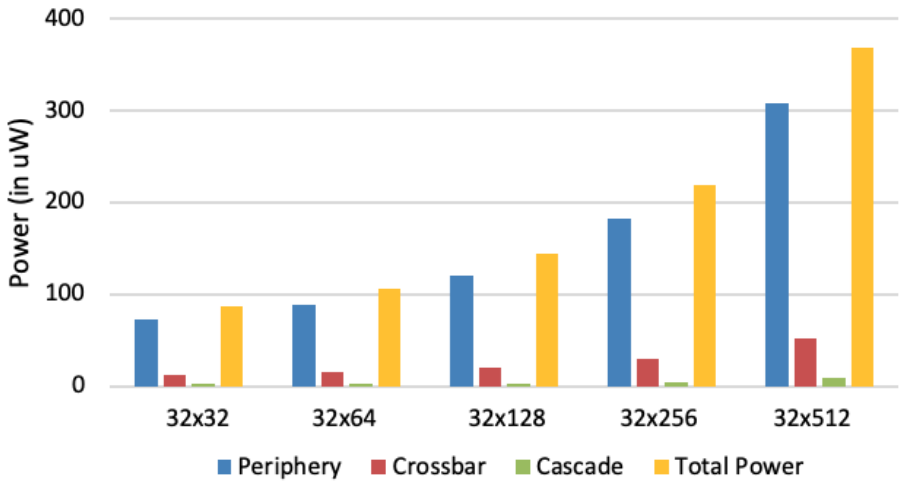

Figure 17: Power share of periphery, crossbar and cascaded logic design as we increase the size of the database (with fixed number of rows of 32) from 32x32 to 32x512 STT-MRAM-based implementation.

4. Optimized CIM-tile with STT-MRAM macro for matrix-matrix multiplication

In this Section, the main idea is to execute CIM-based matrix-matrix multiplication (MMM). Primitive computational unit classified as CIM-P hybrid is extensively explored to perform matrix-matrix multiplication (MMM) with limited (up to 8 bit) operand sizes for neuromorphic and edge-AI applications. Potentially, this is also usable for other multiply-accumulate type operations in other signal and data processing applications.

4.1 Preliminaries for CIM-based MMM computations

Figure 18 describes the fundamentals of using CIM-based MMM unit. The matrices or the operands are mapped to voltage (V) and conductance (G) values.

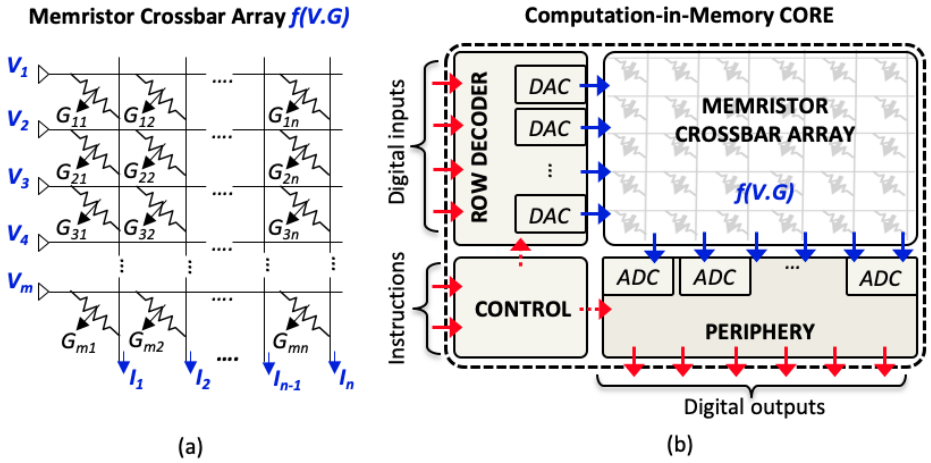

Figure 18: a) Memristor-based crossbar array and, b) CIM Core for MMM computations.

Crossbar array

In **Figure 18a**, a subset of MMM is shown vector-matrix multiplication (VMM) performing several multiply-accumulate (MAC) operations that encompasses the most fundamental computational unit in different domains such as complex neural networks.

VMM is performed by applying a voltage vector $V=V_j$ (where, $j \in \{1, m\}$) to memristor-crossbar matrix of conductance values $G=G_{ij}$ (where, $i \in \{1, n\}$, $j \in \{1, m\}$). At any instance, each column performs a vector-vector multiplication (VVM) or a MAC operation, with the output current vector I, in which each element is $I_i=\sum V_j \bullet G_{ij}$. Note that all n MAC operations are performed with O(1) time complexity.

Periphery Circuit

A CIM core can be inherited from standard well-established memory units such as SRAMs and DRAMs, but with some major modifications to accommodate analog-based computing, as shown in Figure 8b. Firstly, the CMOS-based bitcell comprising the memory unit is replaced by memristor-based bitcell configured in a compact crossbar array, as described previously. The circuit blocks comprising the periphery that supports the bitcell array are significantly modified depending on the operations CIM should accommodate. For MMM operations, the following is needed: 1) Row-decoder becomes complex as CIM involves enabling several rows in a single computation cycle. Also, 1-bit row or word-line drivers are now replaced by digital-to-analog converters (DACs) that convert multi-bit VMM operands into an array of analog voltages. 2) Column periphery circuits performing read operations i.e. 1-bit sense-amplifiers are now replaced by analog-to-digital

converters (ADCs) to quantify currents as digital bit-streams. Post-processing circuits such as shift-and-add are required for MMM 3) Control block needs to deal with complex instructions such as handling intricacies of multi-operand VMM operations as opposed a simple read or write instruction in SRAMs.

MMM Computing

Figure 19 shows the MMM performed between V (voltage) matrix and G (conductance) matrices, each of 128 elements and having 8-bit information/element.

$$\begin{bmatrix} V^8_{1.1} & V^8_{1.2} & \cdots & V^8_{1.8} \\ \vdots & & & \vdots \\ V^8_{127.1} & V^8_{127.2} & \cdots & V^8_{127.8} \\ V^8_{128.1} & V^8_{128.2} & \cdots & V^8_{128.8} \end{bmatrix} \begin{bmatrix} G^8_{1.1} & G^8_{1.2} & \cdots & G^8_{1.8} \\ \vdots & & & \vdots \\ G^8_{127.1} & G^8_{127.2} & \cdots & G^8_{127.8} \\ G^8_{128.1} & G^8_{128.2} & \cdots & G^8_{128.8} \end{bmatrix} = \begin{bmatrix} I^8_{11} & I^8_{12} & \cdots & I^8_{18} \\ \vdots & & & \vdots \\ I^8_{71} & I^8_{72} & \cdots & I^8_{78} \\ I^8_{81} & I^8_{82} & \cdots & I^8_{88} \end{bmatrix}$$

where, $I_{c1} = V^8_{c.1} * G^8_{1.1} + V^8_{c.2} * G^8_{2.1} \ldots + V^8_{c.128} * G^8_{128.1}$;

$I_{c2} = V^8_{c.1} * G^8_{1.2} + V^8_{c.2} * G^8_{2.2} \ldots + V^8_{c.128} * G^8_{128.2}$;

…

$I_{c8} = V^8_{c.1} * G^8_{1.8} + V^8_{c.2} * G^8_{2.8} \ldots + V^8_{c.128} * G^8_{128.8}$ and c is the cycle number

Figure 19: Matrix-matrix multiplication of V (8x128) and G (128x8) matrices with I (8x8) as the output matrix.

The above MMM is performed in 8 cycles as shown in **Figure 20**. In each cycle, the voltage vector of size 1x128 is transposed and applied to the crossbar as row voltages. The current vector is received by the series of ADCs to convert it into 8-bit digital outputs.

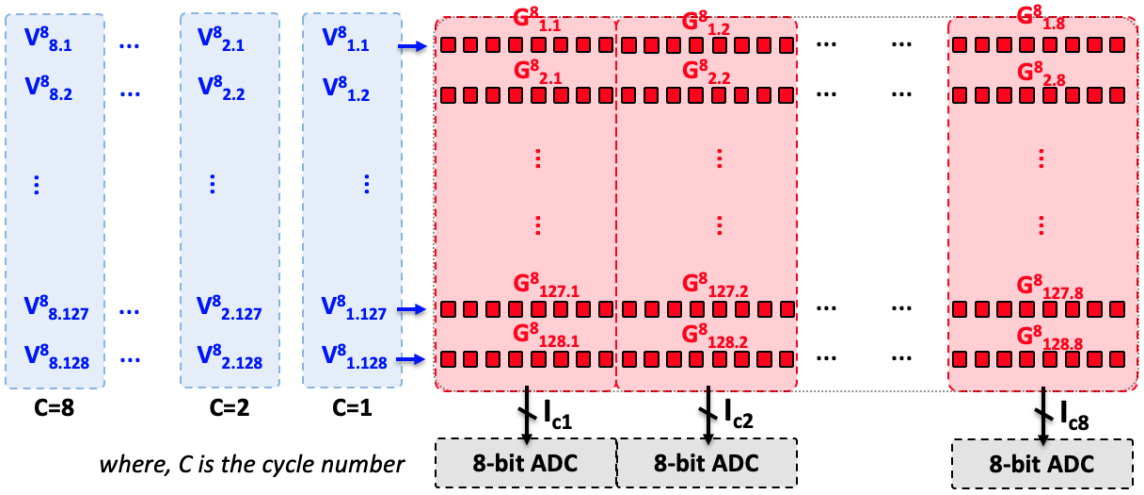

Figure 20: Eight cycles to perform MMM operation as the row voltages are applied in time-complexity manner.

4.2 Simulation Results

Simulation setup and parameters used are same as described in **Table V** in the previous section. In **Table VII**, we summarize the results while performing 8-bit MMM operations with each matrix of element size of 128. We have used proprietary information from IMEC's AIMC design to arrive at these global energy consumption figures, and the details are confidential, so they are not disclosed in this public report.

In each cycle, 128 (=rows) times 8 G^8 operands = **1024 8-bit multiplications and additions** are performed using 8 8-bit ADCs, which consumes **4800 fJ** of energy [7, 8]. This implies that each ADC is performing 128 multiplications and additions (256 fundamental arithmetic operations) while

consuming roughly **600 fJ** and therefore each operation consumes **2.34 fJ/Operation** or **426.4 TOPS/W**. Based on our simulations, ADC consume nearly 20% power and therefore, the overall energy-performance efficiency metrics are **11.7 fJ/Operation** or **85.3 TOPS/W**.

Table VII: Results for STT-MRAM to run a general-purpose MMM of matrices sizes 8x128 and 128x8.

Params (STT-MRAM)	1 VMM Operation (or eight 8-bit MACs)	1 MMM Operation (or eight 8-bit VMMs)
Columns/8-bit ADC	8	
Array Size	128x64	
Cycles Needed	1	8
Setup/Control Delay (ns)	2.5	20
Crossbar Delay (ns)	7.5	60
ADC Delay (ns)	10	80
Total Delay (ns)	**20**	**160**
Total Power (uW)	**1.2**	
Total Energy (pJ)	**24**	**192**
Total Arithmetic Op	**256x8 = 2048**	**256x8x8 = 16384**
Energy/Arithmetic Op	**192 pJ/16384 = 11.7 fJ**	
GOPS	**102.4**	
TOPS/W	**85.3**	

Promising results are obtained using STT-MRAM device technology, with **85.3 TOPS/W** and **102.4 GOPS**, where an arithmetic operation corresponds to one 8-bit multiplication or addition. In other words, **11.7 fJ** per 8-bit arithmetic operation.

5. Integer Matrix-Matrix multiplication

In **WP1**, it was identified that the matrix-matrix multiplication (MMM) is a common operation in the targeted applications. A commonly used data type is either the floating-point or the integer datatype. In **WP4**, it was established that the memristor array can efficiently perform the bit-wise dot product or a dot-product with memristor cells storing a limited number of bits. This resulted in investigations to adapt an application to utilize such dot product operations and to have the host merge the results from the CIM array into integer MMM results - the integer data type is usually easier to implement than floating-point datatypes. Consequently, in the following, we will propose an approach to store multiple bits of an integer number into the memristor array and perform the additional steps necessary to perform an integer MMM operation. The addition operations will be partially performed in the analog array and partially in the digital periphery. Furthermore, the activation of multiple rows of the crossbar can be limited by technology or the resolution of the ADC. Therefore, additional circuitry is necessary to combine the partial sums from the analog array.

In the following sections, an addition unit (in the digital periphery) with a structure tailored for the crossbar array is proposed to aid a crossbar in performing additions targeting integer MMM. The proposed design utilizes minimum-sized adders and is customizable to support varying numbers of ADCs. In the following, first, the mapping of data to the crossbar is discussed. Subsequently, the digital processing which has to be performed on the output of the crossbar to prepare the final result of MMM is presented. This processing has to be done in several steps, which depend on the number and precision of the ADCs used in the CIM tile. We will elaborate on different scenarios to see what changes are required for different scenarios.

5.1 Integer data mapping

In the following, we will utilize a simplified MMM example to highlight the mapping of the multiplicands in the crossbar. We assume that the calculation of a single element $z1$ (in the result matrix) is as follows: $a * j + b * m + c * p$. All values are assumed to be 3 bits in this example. The values *a*, *b*, and *c* are the multipliers and *j*, *m*, and *p* are the multiplicands. Furthermore, we assume that the multiplicands are needed again in other MMM operations and this is the reason to map them into the crossbar since it avoids multiple writes to the crossbar**. Figure 21** depicts the multiplication (for *z1*) written out in full and the mapping of the multiplicands to the crossbar assuming that each cell can only store a single bit. It should be clear from the figure that we first multiply the zero-th bit of the multipliers with the multiplicands and sum them up as they pertain the same weight (20) in the final result. The same multiplicands (indicated in yellow) are needed again when multiplying them with higher-order multiplier bits. In case the crossbar has more columns, the multiplicands *k* and *l* can also be mapped. Otherwise, different tiles are needed to map those multiplicands.

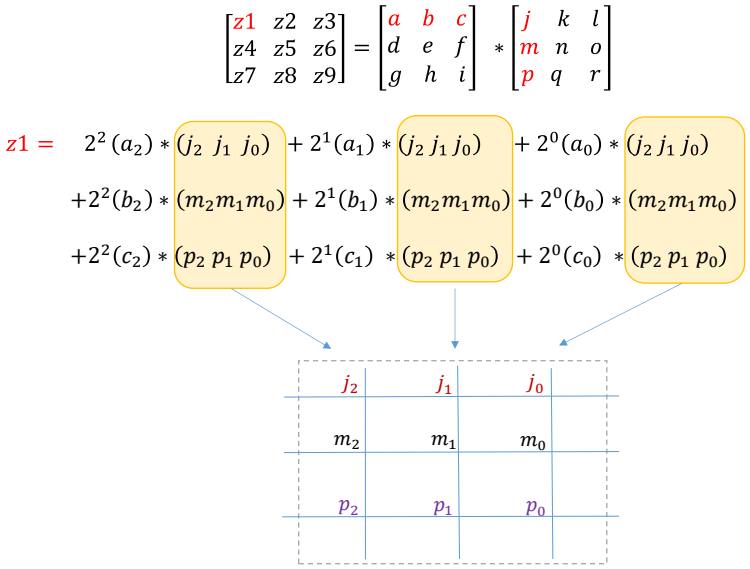

Figure 21: Mapping of integer number into the crossbar

5.2 Proposed organization for addition units

Our proposed adder design clearly distinguishes three adder stages to perform the MMM operation. The intricacies and design considerations of each stage are described in the following.

Stage 1) Adding one column

In this stage, the addition within a single column is performed - indicated by the orange/brown arrow(s) in **Figure 22**. Essentially, the addition in this stage is performed in an analog manner - using Kirchhoff's Law - if and only if (1) the analog-to-digital converter (ADC) is capable of distinguishing between all the possible current levels and (2) all the necessary rows (related to the multiplier) can be activated at the same time - this is dictated by the utilized technology. If either (or both) of the mentioned conditions is not met, the addition needs to be broken down to only adding those numbers of rows that can be supported by the ADC accuracy or technology. This means that the analog addition is only performed among a smaller number of rows and the resulting intermediate results need to be summed up together in the digital domain. The latter is depicted in **Figure 23**. It should be noted that the size of the digital adder here is determined by the largest possible results after all rows have been added - worst case: $\log_2(rows)$. The result of this stage is the summation of all the terms related to one bit-position of the multiplier and one bit-position of the multiplicand.

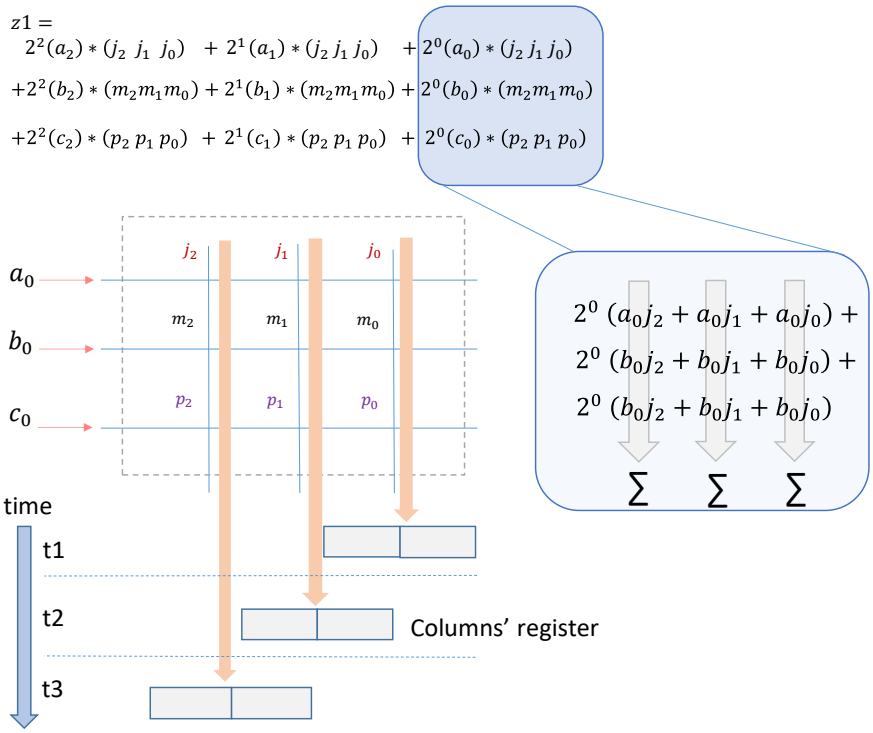

Figure 22: Analog addition inside the crossbar

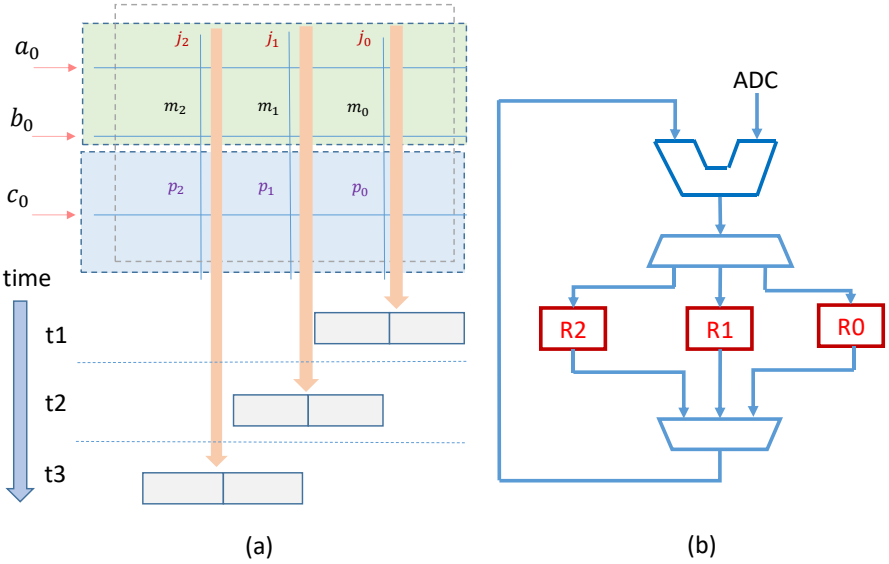

Figure 23: Analog addition with limitation on the number of rows to be activated

Stage 2) Adding multiple columns

As the result of the first stage relates to the summation of only a single bit-position of the multiplicand, the second stage adds up all the terms (i.e., all bit positions) related to the multiplicand (multiplied with only one bit-position of the multiplier) - see the blue box in **Figure 2**. In this stage, we are assuming that the multiple columns are sharing one ADC and that (for now) the integer word size (of

multiplicands) does not exceed this number. The first assumption is not a restriction as all operations described in this stage remain the same if it does not hold. The second assumption is a restriction as loosening it would require additional logic to combine the (intermediate) results. Since this would severely complicate the introduction of our approach at this stage, we defer it to a later section in this paper. The results of each column (see **Figure 24**) relate to different weights of the multiplicand. Consequently, we can use the same adder (maximum size: $\log_2(rows)$) to perform addition as long as each time the addition is performed, the intermediate result is shifted by one position. This means that the lowest significant bit, which is shifted out, can be stored in a temporary register ($R2_{temp}$). The higher-order bits are stored in the register ($R1_{temp}$)– this register must be initialized to zero before the MMM operation is started. The aforementioned addition can be performed while the ADC is "scanning" the columns. The result of the second stage is a partial result of the MMM operation that relates to a single bit-position of the multiplier, e.g., a_0, b_0, and c_0.

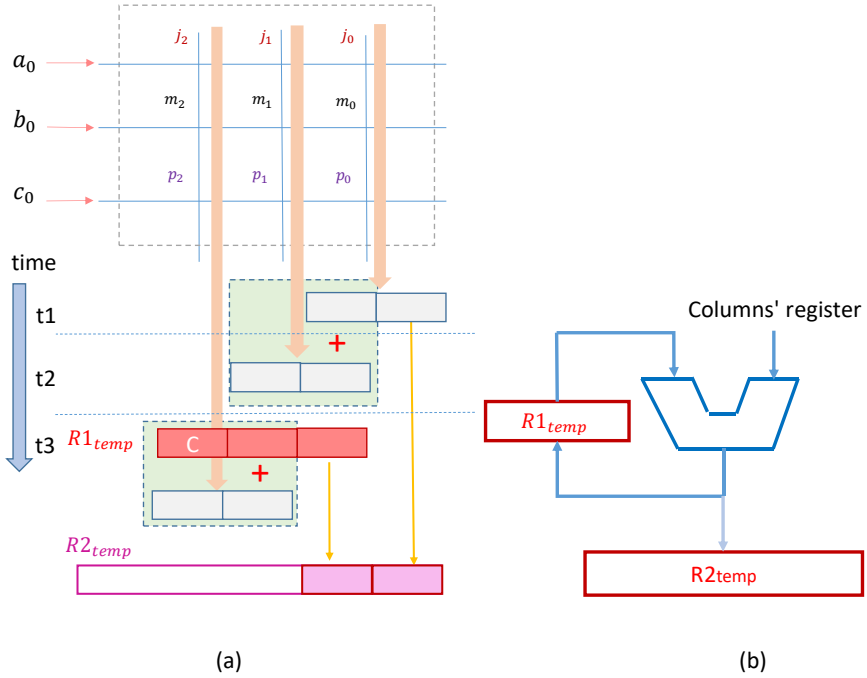

Figure 24: Summation of columns' value in digital domain

Stage 3) Adding higher-order results

In this stage, the partial sums related to different bit positions of the multiplier need to be summed up. This is depicted in **Figure 25**. The partial sums related to the 0-th bit position of the multiplier (gray box) need to added to the partial sum of the first bit position of the multiplier (blue box), and so on. Here, we can employ the same adder structure as described in stage 2 - see **Figure 24**. Only now, the adder and needed temporary registers are larger. **Figure 25** puts together all the hardware discussed before for each processing phase. Following is the generalized size of the registers used in the proposed organization. The size of the adders is equal to their corresponding registers.

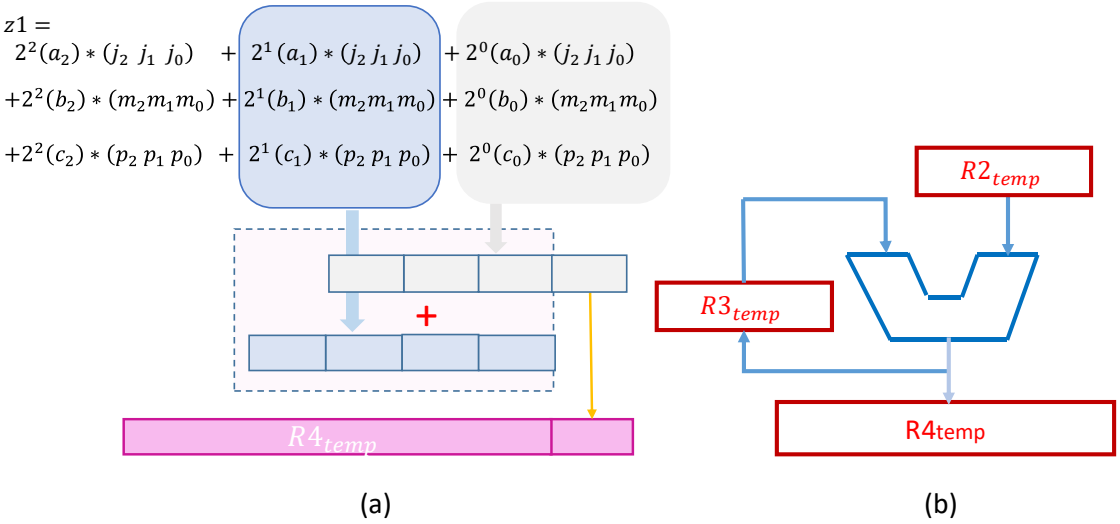

Figure 25: Summation of different bit position of multiplier

Figure 26 puts together all the hardware discussed before for each processing phase. Following is the generalized size of the registers used in the proposed organization. The size of the adders is equal to their corresponding registers.

$$R0 = R1 = R2 = R1temp = log2(crossbar\ height) + log2(cell\ levels) \quad (1)$$

$$R2temp = R3temp = int\ size(multiplicand) + log2(crossbar\ height) \quad (2)$$

$$R4temp = int\ size(multiplier) + int\ size(multiplicand) + log2(crossbar\ height) \quad (3)$$

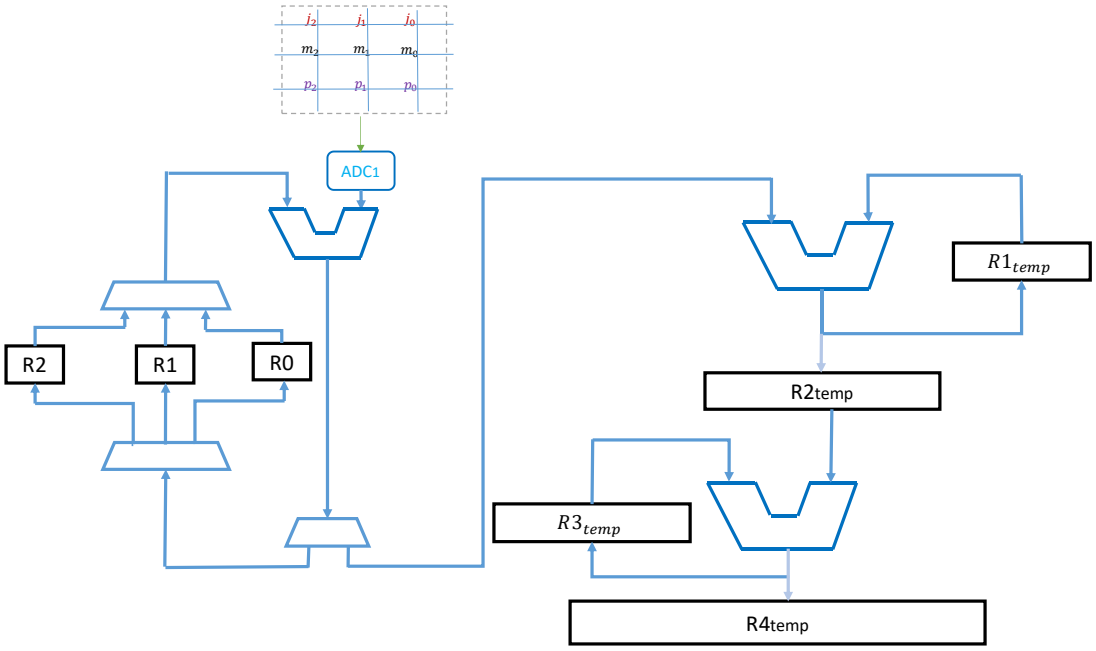

Figure 26: The overall organization required per ADC

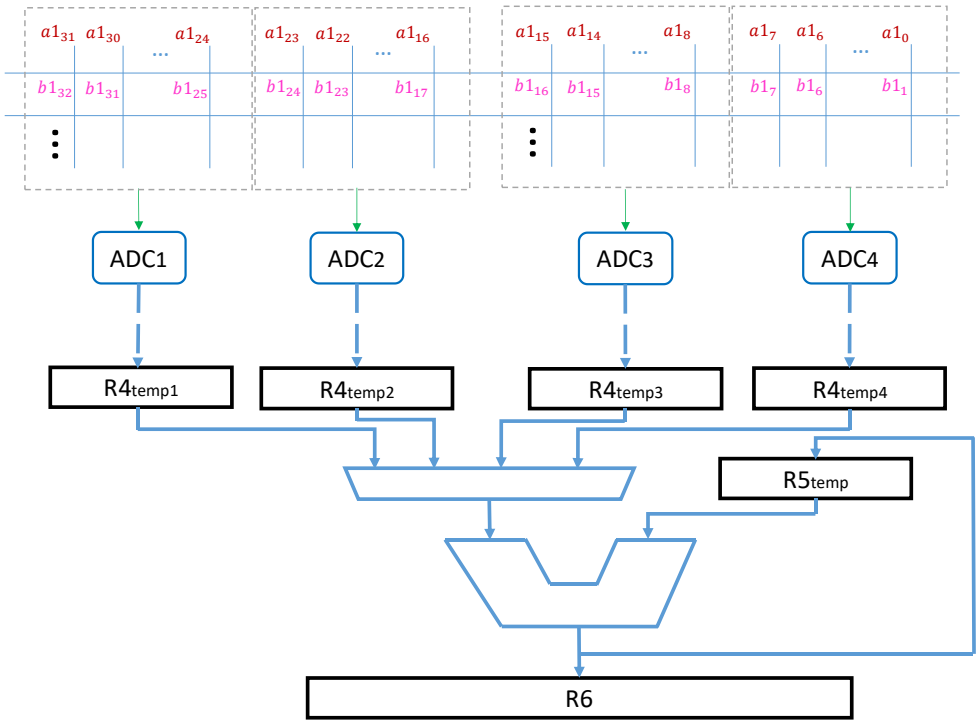

Figure 27: The possible organization required between ADCs

The proposed addition unit till now assumed that the integer size (in bits) equals the number of columns that the ADC is connected for read-out. However, two other possible scenarios must be considered. First, by increasing the number of ADCs, a number stored in the crossbar might be split and distributed over multiple ADCs. An example of storing a 32-bit integer value (covering 32 columns) that needs to be read-out by 4 ADCs is depicted in **Figure 27**. Accordingly, to obtain the final result, the values stored in all the (R4temp) registers, employed for each ADC, have to be summed up. Although more hardware is required as the number of ADCs increases, the length of the registers and adders employed for processes per ADC is decreased. Second, the integer length is shorter than the number of columns read-out by an ADC. In this scenario, either additional control logic must be added to halt the addition at the appropriate moment or the generated nano-instructions sequence should contain take this into account.

Finally, in this section, the structure of the proposed addition scheme was presented to see how the raw data received from ADC processed to obtain the final result of the integer MMM operation. This structure was integrated into our tile level simulator to evaluate this part on the operation/kernel level in terms of energy and performance. The design was synthesized using Cadence Genus and the power as well as latency numbers were imported to the simulator. The simulation results regarding the evaluation of this scheme are presented in **Deliverable 4.7.**

6. Conclusion

The work presented in the different sections of this report resulted not only in selecting hardware design choices for optimizing the memristor based CIM blocks, but, most importantly, the different simulations also corroborate the potential energy advantages of the memristor-based CIM concepts:

In **Section 2** (investigating the impact of the memristor array architecture and ADC design choices on the performance of MAC operations for ReRAM based memristor devices) it was found that, using the best ADC design, the energy for one 8 bit VMM = eight one bit operations, where **energy/(bit op) is between 2 fJ and 30 fJ**. Here, the largest part of the energy consumption is still in the ADC.

In **Section 3** (reporting on the optimization of a CIM tile, based on an STT-RAM memristor array, for performing binary logic operations), it was found that an energy of **26 fJ per bit-wise logical operation is consumed**.

In **Section 4** (reporting on the optimization of a CIM tile, based on an STT-RAM memristor array, for performing Matrix-Matrix Multiplication operations),promising results are obtained, with **85.3 TOPS/W** and **102.4 GOPS**, where an arithmetic operation corresponds to one 8-bit multiplication or addition. In other words, **11.7 fJ** per 8-bit arithmetic operation

With respect the evaluation of the scheme for implementing integer MMM (Section 5), the simulation results regarding are presented in **Deliverable 4.7.**

7. References

[1] T. Chou, W. Tang, J. Botimer, and Z. Zhang, “Cascade: Connecting rrams to extend analog dataflow in an end-to- end in-memory processing paradigm,” in Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, 2019, pp. 114–125.

[2] W. Li, P. Xu, Y. Zhao, H. Li, Y. Xie, and Y. Lin, “Timely: Pushing data movements and interfaces in pim accelerators towards local and in time domain,” arXiv preprint arXiv:2005.01206, 2020.

[3] 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), 2016, pp. 14–26.

[4] S. Yin, X. Sun, S. Yu, and J.-s. Seo, “High-throughput in-memory computing for binary deep neural networks with monolithically integrated rram and 90nm cmos,” arXiv preprint arXiv:1909.07514, 2019.

[5] P.Chi,S.Li,C.Xu,T.Zhang,J.Zhao,Y.Liu,Y.Wang,andY.Xie,“Prime:Anovelprocessing-in-memoryarchitecturefor neural network computation in reram-based main memory,” in 2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA), 2016, pp. 27–39.

[6] L. Song, X. Qian, H. Li, and Y. Chen, “Pipelayer: A pipelined reram-based accelerator for deep learning,” in 2017 IEEE International Symposium on High Performance Computer Architecture (HPCA), 2017, pp. 541–552.

[7] https://www.researchgate.net/publication/339269545_Towards_10000TOPSW_DNN_Inference_with_Analog_in-Memory_Computing__A_Circuit_Blueprint_Device_Options_and_Requirements

[8] https://www.imec-int.com/en/articles/imec-and-globalfoundries-announce-breakthrough-in-ai-chip-bringing-deep-neural-network-calculations-to-iot-edge-devices