Institut für Kommunikationstechnologien und eingebettete Systeme

RWTH AACHEN UNIVERSITY

The present work was submitted to

Institute for Communication Technologies and Embedded Systems (ICE)

RWTH Aachen University

Prof. Dr. Haris Gačanin

Prof. Dr. rer. nat. Rainer Leupers

Bachelor's Thesis SSS-B-034

Evaluation of data representations for a memristor-based matrix-vector-multiplication engine

Bewertung von Datenrepräsentationen für eine Memristor-basierte Matrix-Vektor-Multiplikationsmaschine

by

Stefan-Daniel Vilceanu

Matr.-No. 407531

November 2022

Supervisors:

Prof. Dr. rer. nat. Rainer Leupers

M.Sc. Jose Cubero

Zentrales Prüfungsamt/Central Examination Office

Eidesstattliche Versicherung
Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)
Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/ Masterarbeit* mit dem Titel
I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting) erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt. Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich, dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.
independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:
Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt
Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft.
Para. 156 StGB (German Criminal Code): False Statutory Declarations
Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.
§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt
(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.
(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtigt. Die Vorschriften des § 158 Abs. 2 und 3 gelten entsprechend.
Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence
(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not exceeding one year or a fine.
(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2) and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:
I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Contents

1 Introduction

Motivation

With the last decades' complex computational tasks, it became harder and harder to achieve efficient solutions using classical computing based on the Von-Neumann architecture. Thus, improvement was needed to efficiently continue upgrades in technology.

The In-Memory Computing architecture was created to build efficient hardware in order to represent artificial neural networks [1]. In contrast to the Von-Neumann architecture, where the processing unit and the memory are separated by a shared bus, one of many new designs uses a more recently discovered, nonvolatile, two-terminal resistive switching device, called memristor, depicted in Figure 1.1, and when used in a crossbar array, can enable computation in memory. The memristor has low power consumption and a few nanometers in size, which makes fast switching speed possible [2][3], thus making it an excellent choice for in-memory computing. Moreover, due to the resistive switching feature, the memristor can easily change its state. Applying positive voltage at the active electrode (AE) resets the state of the memristor, putting it in a high resistive state (HRS), while a negative voltage at the AE sets the device into a low resistive state (LRS) [4], thus being able to program different levels and making data storage possible.

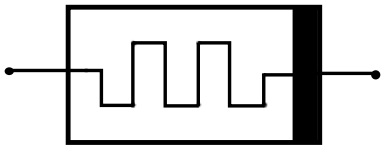


Figure 1.1: Electrical model of a memristor

This thesis studies and evaluates different data representations programmed into a memristor cell for a matrix-vector-multiplication engine. At first, a framework was developed to automatically create SPICE simulation files, being given multiple inputs. The simulation's results from writing, reading, and multiplying data are then processed and evaluated with the help of another script, by looking at the power and energy consumption and the error rate of the interpreted output of the matrix-vector-multiplication.

There are, in total, five chapters, including the introductory part. The second chapter dives into the theory on which this research bases its foundations on, by describing three data representations, the principles of the hardware used, and the process of matrix-vector-multiplications inside a crossbar array. Following further in the thesis, the third chapter describes the framework with its python scripts used in this work by identifying key steps

and methods of the process. Next up, in chapter four, an evaluation of the results gathered using the last section's implementation for all the data representations is shown, by looking at multiple characteristics, such as error rate, matrix-vector-multiplication interpretation, and power and energy consumption of a cell and crossbar. Last but not least the conclusion of this thesis is presented in the final fifth chapter, together with a few suggestions for future work.

Previous work

In the past [5], there have been studies related to the power modeling of memristor crossbars, which focused exclusively on passive crossbars, the memristor being the only component. Binary was the only data representation used, meaning that bit-slicing had to be performed in order to work with values bigger than one. Moreover, the analytical formulas designed to calculate the power usage were not verified with the results of a low-level simulation. This bachelor's thesis aims to extend the passive cell to a 1T1R, enabling multibit programming in two new approaches. Also, the crossbar is to be simulated using Cadence Spectre Simulator in order to prove the correctness of new updated analytical formulas matching the 1T1R cell.

A similar approach to the one performed in this thesis was made in [6]. A model of a dot-product engine was developed using 1T1R cells to simulate a memristive crossbar with high flexibility and rapid matrix-vector multiplications. This model allowed multibit programming; however, only a cell resolution of two bits was tested. This thesis aims to study cell resolutions of up to four bits and to compare their energy efficiency and multiplication accuracy with one another. Moreover, a new multibit mapping algorithm will be tested, which should deliver better programming results according to [7].

2 Theory

2.1 Memristive Crossbar Arrays

Crossbar arrays, or simply crossbars, are analog structures well suited for the use of memristors. They are built out of two electrodes, one at the top and one at the bottom. In between these two wires, there is a cell, which can be passive, shown in Figure 2.1 incorporating a memristor, or active, consisting of a transistor connected in series with a memristor, also named a 1T1R cell [8]. The top electrode is typically called word line (WL), while the bottom electrode is named bit line (BL). In the case of a 1T1R cell, there is also a third connection for the gate of the transistor, called selection line (SL). In order to have a functional crossbar array, there need to be digital-to-analog converters connected to the WL to set operational voltages, while current sensing electronics and analog-to-digital converters have to be installed to the bit lines [6].

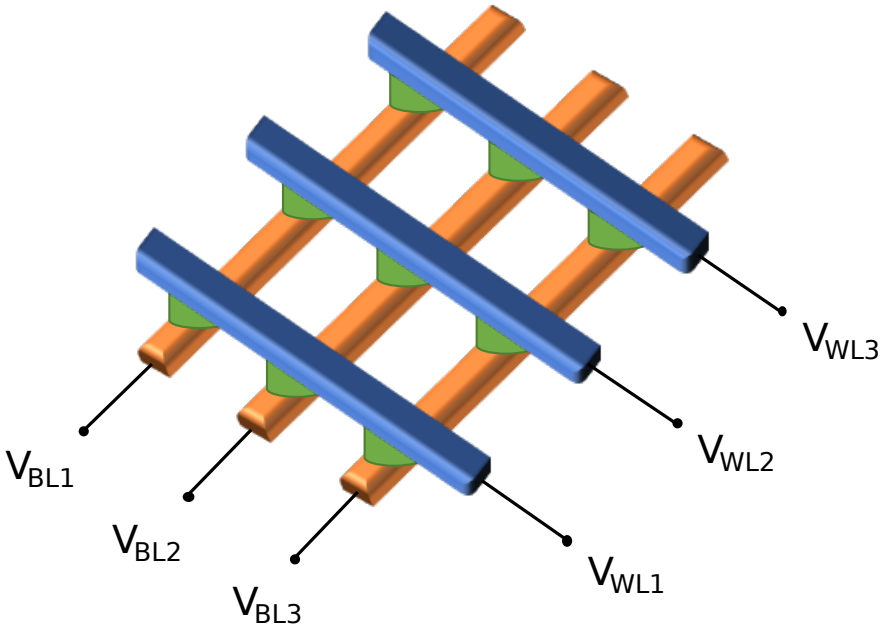


Figure 2.1: 3D model of a passive crossbar array

This thesis will focus strictly on the 1T1R cell, Figure 2.2, as the passive construct was already discussed in a previous work. The 1T1R configuration brings overall a higher energy efficiency thanks to the lack of leakage currents that normally exists in a passive crossbars, in the non-selected cells. This happens because of the transistor, which acts as a voltage-controlled switch. Moreover, the transistor enables new ways of mapping data into the cell.

As described in [7], there the two methods used, the incremental step pulse with verify algorithm (ISPVA) and the incremental gate voltage with verify algorithm (IGVVA). The ISPVA increases step by step the word line voltage while feeding a fixed gate level to the transistor. The IGVVA, on the other hand, has a fixed word line level, while at the selection line, constantly increased voltage pulses are output in the gate. Results show a more accurate conductance programming for the IGVVA. For this reason, this algorithm will be used further.

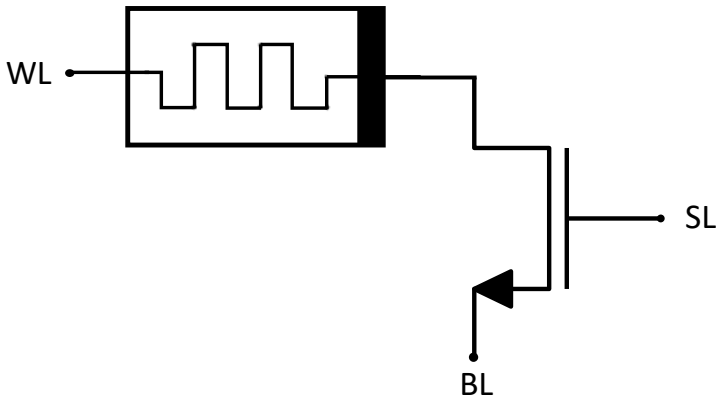


Figure 2.2: 1T1R cell

2.2 Data Representations

To be able to achieve higher levels of energy efficiency, there was the necessity to develop new, more efficient methods of mapping data inside a crossbar before matrix-vector multiplications were executed. In this thesis, there are three data representations being approached, single-bit, multilevel and differential, each one of them having different advantages and disadvantages [9].

2.2.1 Single-Bit

The single-bit data representation, Figure 2.3, is the most common way of writing numbers. It is met today in most conventional computers with digital memories based on the Von Neumann architecture, and it has a straightforward working method. To write a decimal value x into binary, N cells need to be used, where N must fulfill: $2^N - 1 \geq x$. The cell that needs to be written can only represent one bit, which means that it can find itself in only two states, in the high resistive state or in the low resistive state. This method keeps the complexity level lower but has a significantly higher energy consumption.

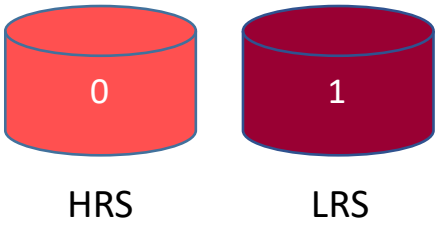


Figure 2.3: Single bit representation

2.2.2 Multilevel

The second data representation approach is the multilevel or the multibit, presented in Figure 2.4. This method can be seen in analog memories, where the device has its conductance divided into 2^M equally distributed levels ranging from G_{min} to G_{max}, M being the cell resolution. Thus, converting a decimal value x, where $x \leq 2^M - 1$, can happen in only one cell instead of N, as in the single-bit case. A problem can appear when M is too large because the equally spaced conductance levels will be too small, thus making the value interpretation very sensitive to errors.

By using the multilevel mapping technique and with a proper choice of M, there is a considerable increase in efficiency and a significant reduction in analog memory devices that need to be used. The downside is a higher complexity by having to interpret more values.

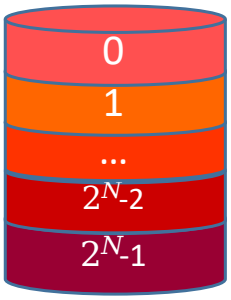


Figure 2.4: Multibit representation

2.2.3 Differential

The differential memory mapping method, Figure 2.5, relies on the implementation of the multilevel approach. The critical difference is given by the ability to map positive as well as negative numbers in the crossbar, these ranging between [-2^N + 1, 2^N - 1], where N represents the cell resolution. In order to achieve this aspect, there needs to be a pair of two memory devices instead of only one. The data is being mapped, just as in the multilevel approach, in both of the cells, and by using a differential amplifier, the desired value is interpreted as the subtraction of the currents from the two cells [10]. This results in having a crossbar with double the columns, which means twice as many memory devices, but, in return, there are (2^{N+1} - 1) available values, and it also makes mapping of negative numbers possible.

2.3 Representing Numbers in a Crossbar

When working with the above-mentioned data representation, additional steps need to be performed depending on what data needs to be programmed in the crossbar. There are two cases when extra preparation is required. Firstly, when the to-be-written numbers are bigger than what the cell resolution can map, bit-slicing is needed. For example, if the decimal number 11 needs to be programmed, the single-bit representation would require four cells/slices, and multibit with a cell resolution of 2 would require two. Only in the scenario of multibit with $N = 4$ or differential with $N = 4$, bit-slicing would not be needed. The second case addresses the representation of negative numbers, where a bias needs to be added to convert them to positive only. This is only needed when working with the single bit or the multilevel representation, as the differential technique can directly program negative values [11] [12].

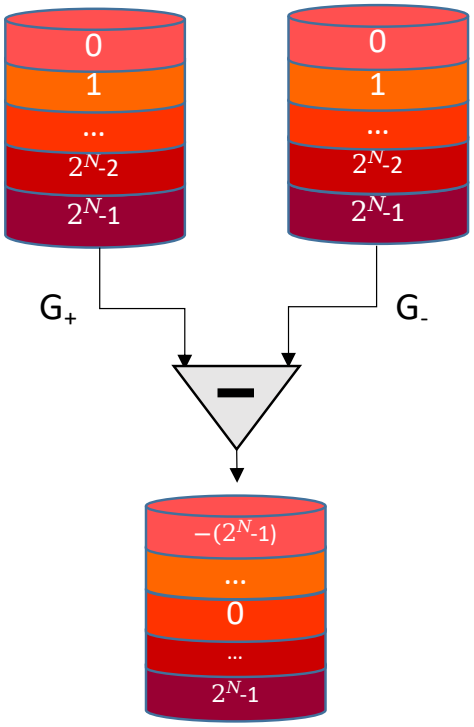


Figure 2.5: Differential representation

2.4 Matrix-Vector Multiplication

The matrix-vector multiplication unit (MVMU) is a vital part of the in-memory computing technique, as it is responsible for the arithmetic operations but also the speed of the device. For a successful matrix-vector multiplication (MVM), there are especially two important factors required: accurately interpreted numbers and a very short multiplication pulse. MVM operations often occur in machine learning workloads, neural networks in particular [13].

The working method of the MVM is presented as follows. The crossbar must already store programmed values, which will represent the matrix. As input, there must be a multiplication command, followed by the actual input data, that activates specific rows from the crossbar. Afterwards, the currents through all the cells from the activated rows will be added up, remaining only to sense the summed current at the bit line. If the transposed crossbar is an (n,m) matrix, the input an (n,1) vector, then the output will be a (1,m) vector.

Figure 2.6 presents an example of MVM, where a (3,1) sized vector input activates rows 1 and 3 from the 3x3 matrix. The values in the same column and from enabled rows are then summed up at the end of the respective column, which outputs a (1,3) vector. Figure 2.7 shows the applied voltages in the crossbar during this MVM example.

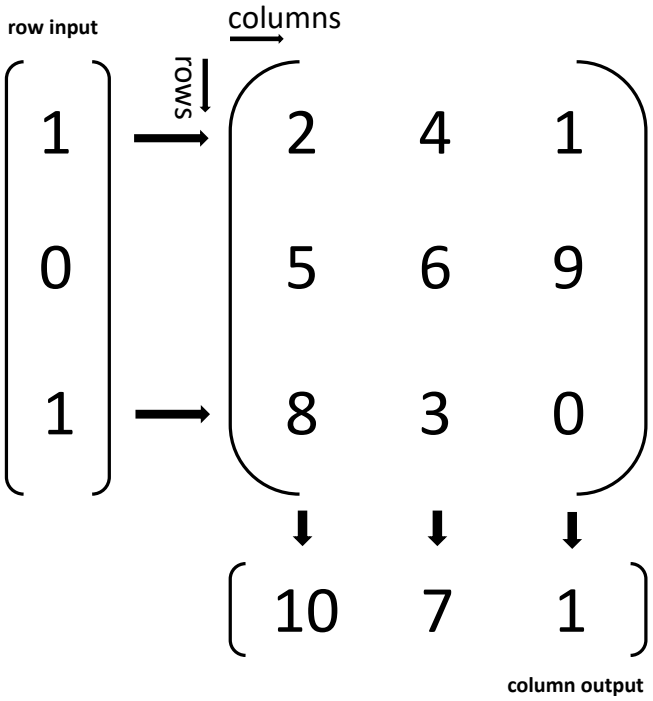


Figure 2.6: Example of MVM

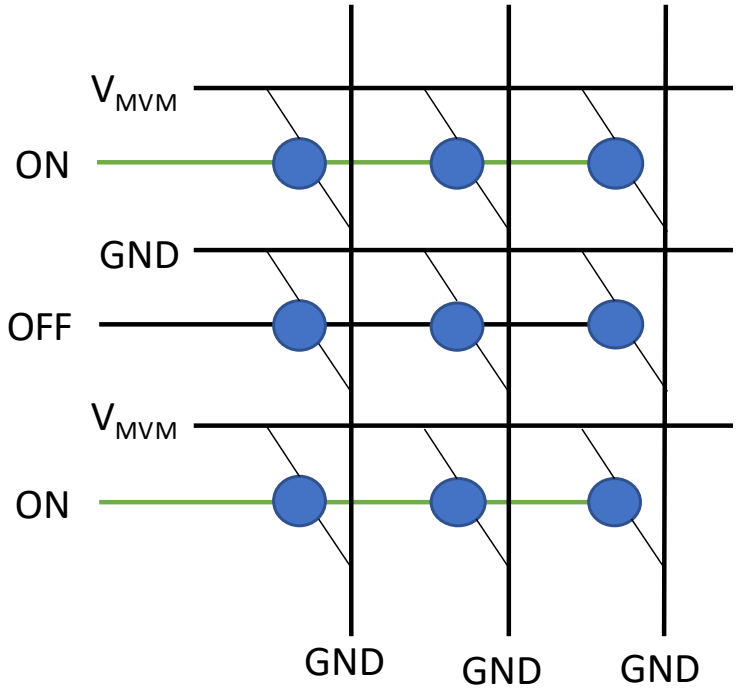


Figure 2.7: Crossbar during an MVM

3 Framework

The framework of this thesis is divided in three hierarchical blocks. In order to evaluate different data representations, crossbars needed to be simulated. The first and main block is represented by an object-oriented python script that writes netlist files, which afterwards can be simulated using Cadence Spectre Simulation Platform. The second part relies on interpreting the write and MVM results of the simulation but also on determining the error made by the interpretation, which is done using another script. For the last segment of this work, a final python implementation calculates the power and energy consumption of a cell and of the whole crossbar.

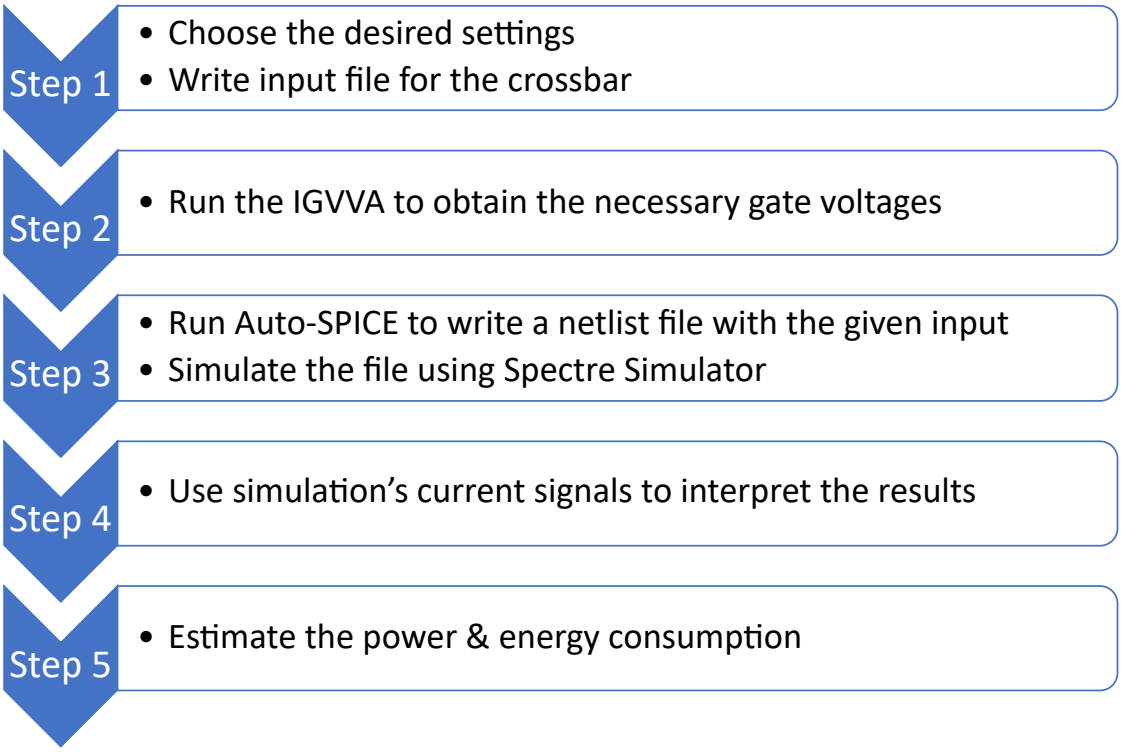


Figure 3.1: Workflow

Figure 3.1 shows the main steps of the process, from setting up the crossbar and until calculating the results.

3.1 Auto-SPICE

The memristor crossbar netlist generator script, called Auto-SPICE, was initially started by a fellow ICE student, but for the purpose of this thesis it had to be further engineered by adding more features and improving its efficiency.

3.1.1 Input Files

Auto-SPICE starts with a Comma-Separated Values (CSV) input file for the desired crossbar. Here, the number of rows and columns, the cell resolution, and a total maximum of four unique action commands can be specified. The **read** command can be inputted in two ways. The first one allows specific cells, from specific rows, from the crossbar, to be read, and the rest remain idle, while the second option can directly activate all the cells from specified rows. The **mvm** command is similar to the last described action, as it is itself a type of read that also allows row specific activation to perform the matrix-vector multiplication. The main difference here is that the **mvm** happens for all the rows simultaneously, rather than each on a different time stamp, thus making the concept of MVM possible. Next up, the **write** command is also split into two, a set and a reset. In this thesis, before setting values in the crossbar, the cells need to be reset (be in the HRS). The reset-**write** action is a cell-specific command, which allows cells from certain rows and columns to be reset, but usually, the full crossbar is selected. The set-**write** is also cell-specific. For the multibit representation, depending on the unit resolution N, it can accept inputs between [0, 2^N - 1], and based on these numbers, the gate voltage of the transistor is accordingly selected.

Listing 3.1 shows an example of a 4x4 crossbar input file, where the first column represents the operation and the second one, in case of cell-specific instructions, is the selected row. At first, all the cells are being read and afterwards, the programming takes place with a cell resolution of two, mapping values between 0 and 3, followed by the second type of read, in line 9, where a 1 represents reading a whole row, rather than a specific cell. This principle is taken further for the two MVMs, in lines 10 and 11. The first multiplication only activates the first and third rows, while the second one activates everything.

Listing 3.1: Example of a 4x4 crossbar input file with multibit

```
1.  r,0,1,1,1,1
2.  r,1,1,1,1,1
3.  r,2,1,1,1,1
4.  r,3,1,1,1,1
5.  w,0,1,3,2,0
6.  w,1,2,3,0,0
7.  w,2,1,3,2,1
8.  w,3,3,1,1,2
9.  readrow,_,1,1,1,1
10. mvm,_,1,0,1,0
11. mvm,_,1,1,1,1
```

Figure 3.2 shows the simulated current in the first BL from the given instructions in Listing 3.1.

As a second input, simulation, crossbar, transistor, and memristor parameters are specified in a JavaScript Object Notation (JSON) configuration file. The more important ones are, for example, input type, the Verilog models for the used components, type of the simulation, pulse time length, different for read/set and reset commands, time period between pulses, maximum step time for the simulation, as well as multiple voltages for each operation for the WL, BL, and SL. The input type is the main parameter of this script as it describes the data representation that will be used in the simulation after the netlist file is written. The types are **parallel binary operation**, **parallel multibit operation**, and **parallel differential operation**. All these three have a way of doing pulse parallelization in order to have a more efficient simulation. The read command is performed at the same time, under a single pulse for every cell of the row in all three representations, as the read

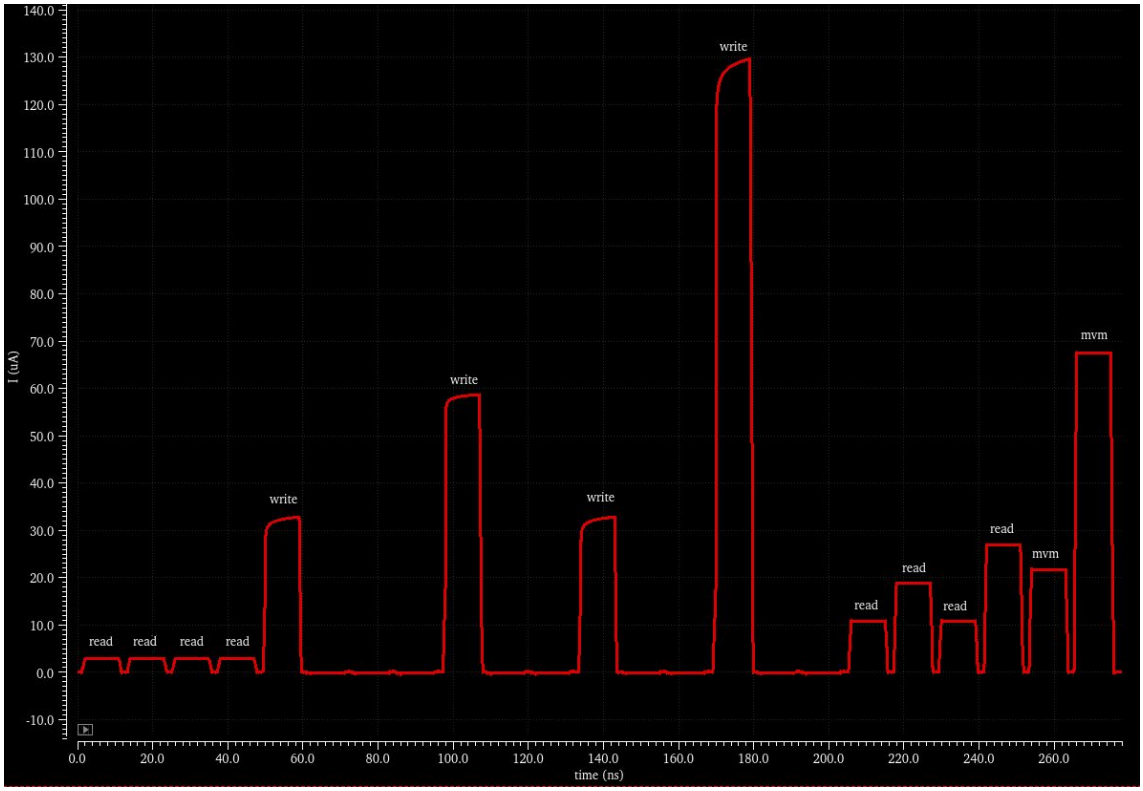


Figure 3.2: Current signal at the first BL from the given example

gate voltage always remains constant. For the write operation, the concept becomes more complex, as this command has, in the smallest representation, binary, two operations, set and reset, therefore needing a gate voltage for every single action (two in the case of binary). The parallelization approach here is to gather all the identical inputs from a row and to represent them with a single pulse. For example, if a crossbar has N cell resolution and M columns, it will result in having a maximal number of 2^N different pulses while writing a row instead of having M pulses for every single cell of the row (usually, $M \gg N$).

3.1.2 Operations

Set/Reset/Read Operation

These three cell-specific procedures have a similar working method. Assuming there is a **m** x **n** sized crossbar and an action needs to be performed on cell (i, j), the voltages on the i-th WL and SL need to be set to HIGH, while all other WLs and SLs have to be LOW (GND). In order to have current flowing only in the selected cell, there needs to be a potential difference between i-th WL and j-th BL and no potential difference between i-th WL and all the other BLs. This means that the j-th BL needs to be grounded, while the voltage levels of the rest need to be HIGH. At this stage, because all WLs, excluding the i-th, are LOW, and all BLs, excluding the j-th, are HIGH, there could be current flowing in unselected cells, but due to the grounded transistor gate of all these cells, zero current is allowed to pass.

MVM Operation

The MVM operation is exclusively designed for row input. To do MVM with the whole crossbar, all the WLs and SLs need to be set to HIGH, while every BL has to be LOW. If certain rows are not considered during the MVM, then the WLs and SLs of these corresponding

rows need to be grounded. The result of the MVM is given after the current in each BL is measured, as this is the sum of all the cell currents in the column with an activated row.

3.1.3 Pulse Specifications

Constructing the right pulses for a simulation can be difficult, especially when the cell is not passive but a 1T1R cell. The pulse length has a huge importance, as it basically decides how fast the read/write operations and the MVM should be performed. Initially, the work was done with longer pulses, all operations lasting for 1ms, to ensure that the framework is capable of achieving the desired outcome by building it on more than long enough pulses.

Under these circumstances, proper voltages needed to be tested. [4] used at the WL around 1V for the set, approximately -2V for the reset and 0.2V for the read operation when designing a memristor crossbar. Due to the addition of the transistor, these WL voltages needed to be increased to satisfy the requirements of both components. For the set, 1.5V was chosen. Here, it is important to note that working with too big of a voltage at a transistor's drain can trigger its breakdown region, thus destabilizing the simulation. The reset was adjusted to -2.5V, which had to be low enough to successfully switch the memristor to the HRS while the read was set to 0.3V. Having already established proper voltages for the operations, it was possible to lower the pulse length substantially. Most of the work was done using 10ns (100MHz) pulses for the set, read, and mvm operation, 100ns (10MHz) for the reset, and 1ns pause between pulses. Later experiments showed that the shortest achievable pulses with this setup were, in fact, 5ns (200MHz) for the set/read/mvm and 25ns (40MHz) for the reset. When trying to go even lower in time length than 5-25ns, the simulation had massive and abundant spikes while setting and reading, and during the reset, the memristor could not be fully and successfully switched into the HRS.

3.1.4 Data Representations

In the theory chapter, the three data representations were briefly presented. In this subsection, it will be described how the values are being programmed in the crossbar. For this work, the gate voltage of the transistor plays an essential role in cell programming.

Starting with the **single-bit** representation, only two values can be mapped, 0 and 1. These values can also be represented as HRS and LRS of the memristor, but also as minimum conductance G_{min} and maximum conductance G_{max}, respectively maximum resistance (R_{max}) and minimum resistance (R_{min}) of the cell. For the 1 (set), the calculated gate voltage was 3V, while the 0 (reset) had a 3.15V gate level.

The **multibit** representation, which is also more interesting than the single bit, dives deeper into the method. To achieve multibit mapping, as mentioned in chapter two, there are two possible algorithmic approaches, ISPVA and IGVVA. The target is to have 2^N equally distributed conductance levels to be able to represent 2^N values, given by 2^N different gate voltages, N being the cell resolution. After choosing the algorithm, in this case, IGVVA, there are two more methods before proceeding further, which is the possibility to choose between mapping equally distributed conductance levels for the whole cell (perceiving the series connected memristor and transistor as a single unit) and mapping only the memristor in equally distributed conductance levels. Both approaches were tested in this thesis, with the second one being more complicated due to having to subtract the influence of the transistor when interpreting the numbers. This second method also requires some extra information, such as the voltage at the connection between the two components or, if it is

known that the transistor works in the triode region, its approximated resistance. Overall in this work, the focus was more on the first method, due to its straightforwardness. If the cell resolution has N bits, then there will be 1 HRS and $N-1$ LRSs. It is essential to mention that in this thesis, in order to set values in the crossbar, the cells need to be reset because switching between LRSs is not possible, due to the memristor's construction. Thus, an energy consumption improvement has been developed when using the multibit mapping method. When setting a cell, an input 0 will not do another reset (the cell is already in HRS), but it will perform another set pulse, with the gate voltage being the threshold voltage V_{th} of the transistor. Thus, when writing, the *set 0* pulse consumes a lot less energy than a *reset 0* one, keeps the cell in the same HRS, and when reading or MVM-ing, the current remains the same as if a *reset 0* would have been performed. Because all the other inputs, from 1 to $N-1$ are used to set a cell to the corresponding LRS, and the 0 is also used for setting as just described above, *-0* is now assigned as input to perform the actual reset.

The same multibit mapping concept is applied for the **differential method**, using the same gate voltages for programming as calculated per the IGVVA. As mentioned in the theory chapter, this approach uses double the columns to gain access over negative values by subtracting the paired cells between two columns, thus increasing the number of values that can be written, from 2^N to $2^{N+1}-1$. Using a differential matrix, as the example in Figure 3.3, provides two options when writing cells to be afterwards subtracted. The first one is to always map the lowest resistive state in a cell at position (i, j) and any of the N other states at (i, j+1) to obtain 0 or positive values when subtracting (e.g., LRS3 - LRS2 = 1, for 2-bit resolution). To get negative numbers, at position (i, j) any of the N states needs to be written while at (i, j+1) it must be the lowest resistive state (e.g., LRS1 - LRS3 = -2, for 2 bit resolution). The green-marked squares from the differential matrix show precisely this pattern. The second option is exactly the opposite operation. For positive numbers, the minuend can be any state, while the subtrahend must be the HRS0 (e.g., LRS2 - HRS0 = 2), and for negative numbers, the positions are inverted (e.g., HRS0 - LRS1 = -1), presented in the orange colored squares of the matrix.

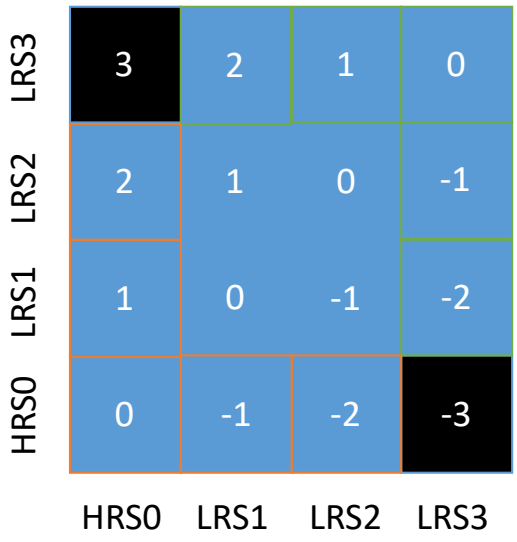
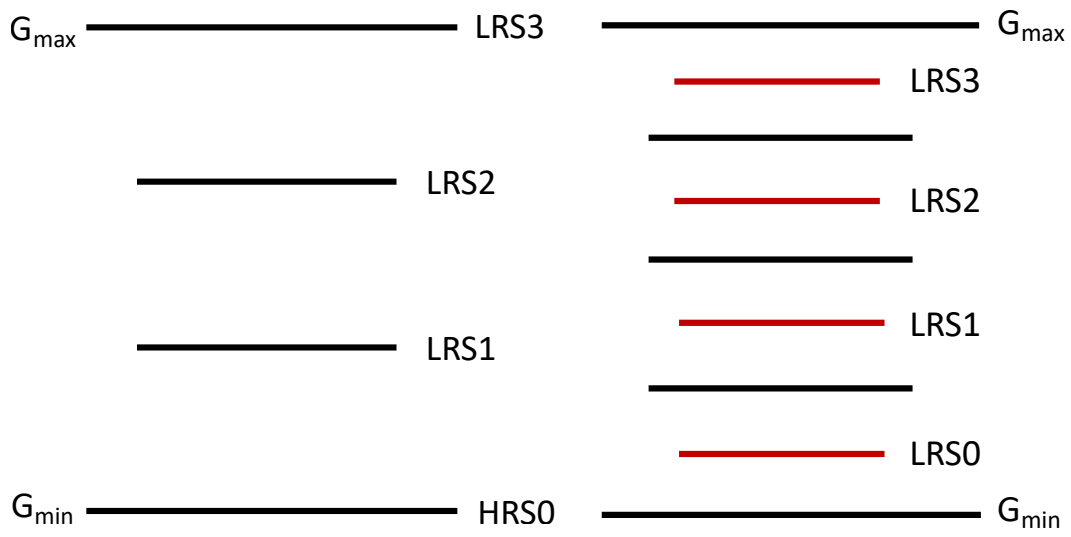


Figure 3.3: Example of differential matrix



(a) HRS0-LRS3 (b) LRS0-LRS3

Figure 3.4: Mapping methods for the IGVVA with $N = 2$

3.1.5 IGVVA

The IGVVA was used in this thesis to help map multibit values in cells. This algorithm was directly implemented in the Auto-SPICE script to generate a corresponding netlist file to ease future experiments with different models. In this work, the memristor used was the JART VCM v1b model [4], while for the transistor, various versions were tested, such as 32nm and 130nm [14][15]. In order to perform the IGVVA, the threshold voltage of the transistor has to be known to be able to perform a more accurate gate sweep. For the 32nm model, the gate voltage was swept from a little under $V_{th} = 0.493V$ and until around 3.6V, where the conductance level was fully saturated. In the case of the 130nm model, the sweep range was between 0.3V and 3.1V. An essential element is the step increment in order to gather as much accurate data as possible. A good choice for this gate voltage step is 10mV. The pulses for the IGVVA are always a pair of set and read, which means after every set operation at gate voltage x, there is a read with a fixed gate, followed by the next set pulse with $V_{gate} = x + 10mV$ and so on.

After the netlist file is written and simulated, specific current and voltage signals are saved in order to later obtain the desired gate voltages with the help of another python script. Firstly, the conductance needs to be calculated by dividing the current by the voltage signal and picking the last measured value from all the read pulses, thus receiving a discrete conductance signal as function of the swept voltage. In order to receive a gate voltage at a specific conductance level, which might be somewhere between two points, the discrete signal needs to be transformed into a continuous function by curve-fitting all the discrete y-axis and x-axis data and afterwards calculating the inverse function. In order to obtain a more accurate continuous curve, a 6-th-order polynomial was used to represent it. Its parameters were calculated using the *polyfit* function of the *numpy* library. Calculating the inverse function was also possible thanks to the strictly monotonic characteristic of the curve and using the function *inversefunc* of library *pynverse*. After the continuous, inverse function was determined, it only remains to choose how to correctly divide the conductance into intervals.

The idea of the IGVVA is to map 2^N equally spaced conductance levels starting from G_{min} (HRS) and until G_{max} (lowest resistive state), depicted in Figure 3.4a, this being the most commonly used method, also being focused in this work's implementation. Another conductance distributing approach, which was also implemented but only briefly studied in this thesis, divides the same conductance range into 2^N intervals. However, instead of starting from the min and finishing at the max value, the chosen levels are precisely in the middle of these intervals, Figure 3.4b, therefore offering a buffer zone at the ends of this range. Moreover, because there is no G_{min} or G_{max} in this second approach, it means there cannot be a true HRS or a true **lowest** resistive state, but only 2^N LRSs. This also means that trying to map the value *0* (LRS0), it would not be equal to the same *0* after a reset operation is performed, which results into having higher energy consumption when writing and reading a value *0* than in the first approach. On the other hand, the LRS($2^N - 1$) tends to be less energy-consuming than the lowest resistive state from the first described method, but all in all, both approaches are valid and worth applying.

After using the first conductance distributing approach, for a 2-bit resolution, the following gate voltages for programming equally distributed conductance levels were calculated:
32nm transistor: 0.493V, 0.891V, 1.265V, 3.006V;
130nm transistor: 0.378V, 0.747V, 0.946V, 2.938V.

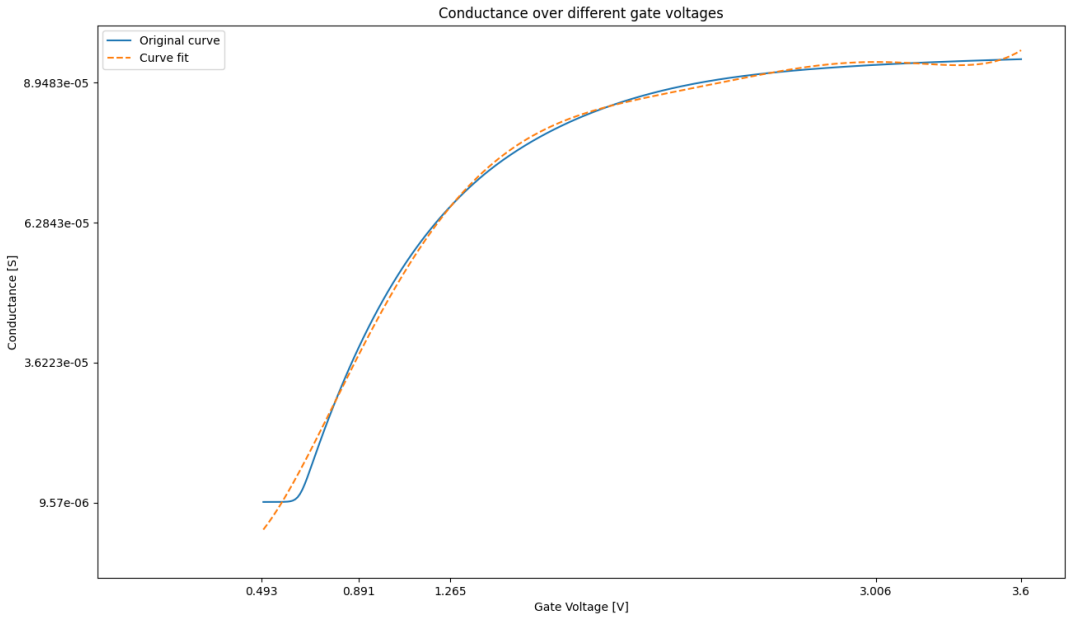


Figure 3.5: Plot of the IGVVA for a 2 bit resolution, 32nm transistor

Figure 3.5 presents a plot with the conductance on the y-axis and the gate voltage on the x-axis. The blue curve represents the discrete data set, starting from G_{min} and ending in G_{max}, while the other line is the continuous curve fit. Although the orange curve follows the original quite closely, in order to have an even better match between the two, the plot was later divided into several sections to make the fitting even more precise due to the very abrupt slope at the beginning.

3.2 MVM-Interpretation

The interpretation represents an important task in this thesis, as a correct evaluation of the programmed values, but especially of the MVM's result, makes the use of this work relevant. This step takes place after the simulation of the Auto-SPICE-generated netlist file by using some key signals from it.

3.2.1 Approach

As a working method, a small script was implemented to interpret the **read** and **mvm** operations, but also to check the correctness of this process. As input, the WL voltage and the BL current signals from the SPICE simulation are needed to perform the task. Additionally, the size of the crossbar, the cell resolution, and the min and max conductance values, in this case, the value of the HRS and of the lowest resistive state, need to be known as well. Afterwards, with a proper formula, the simulation results can be well interpreted.

3.2.2 Interpretation Formula

A formula was developed for each of the two ways of conductance mapping. Starting with the less complex one, which gives better interpretation results because it does not include any approximated parameters and it was also focused more on in this thesis, it bases on the first presented method in 3.1.4 (perceiving the memristor and transistor as a single unit when calculating equally distributed conductance levels). I_{read} is the sensed current in a specific BL, V_{read} the WL voltage of a read operation, G_{min} the smallest conductance level, from where the mapping begins, N represents the cell resolution, and x is the to be interpreted value.

$$x = f(I_{read}) = \frac{I_{read} - G_{min} * V_{read}}{(G_{max} - G_{min}) * V_{read}} * (2^N - 1) \tag{3.1}$$

Equation 3.1 represents the interpretation formula for reading a single cell. It is worth noting, that subtracting $G_{min} * V_{read} = I_{min}$ is essential, because the sensed currents always begin from a specific minimal read current level I_{min}. This means that, e.g., mapping the value 0 (HRS) does not result into 0 μA sensed, but in I_{min}, or mapping a 1 is nothing else but a $\Delta + I_{min}$, which can be confirmed by the $y = mx + b$ equation.

The other formula, for the second, described in 3.1.4, method of mapping equally distributed conductance levels only in the memristor by having to subtract the influence of the transistor afterwards, contains additionally the approximated resistance R_{on} of the MOSFET.

$$x = f(I_{read}) = \frac{V_{read} * G_{min} * \frac{1}{R_{on}} - I_{read} * G_{min} - I_{read} * \frac{1}{R_{on}}}{(G_{max} - G_{min}) * (I_{read} - V_{read} * \frac{1}{R_{on}})} * (2^N - 1) \tag{3.2}$$

Equation 3.2 represents the interpretation formula for reading a single cell by subtracting the transistor's influence. This formula will not be discussed further in this work.

The above equations are designed to interpret the value when a single cell from a given column is activated, which makes it a perfect match for debugging to check if a number was correctly written in a specific cell. On the other hand, to be able to perform MVMs,

which usually means that more than just one single cell from a column is activated, the formulas need to be extended. The minimal sensed current I_{min} needs to be subtracted as many times as rows are being activated when performing the MVM. The updated formula looks as follows:

$$x = f(I_{read}, k) = \frac{I_{read} - k * G_{min} * V_{read}}{(G_{max} - G_{min}) * V_{read}} * (2^N - 1) \tag{3.3}$$

where k is the number of activated rows.

However, last described formula raises the following question: How do the peripherals know how many rows are being activated during the MVM in order to subtract that many times I_{min} from the total current in the BL. There are two approaches further, with the first one simply not subtracting $k * I_{min}$, thus leaving the small error be. For very small crossbars, it would still be possible to interpret the data correctly, but all in all, it is not an ideal method. The second approach takes its similarities from the differential data representation, as it requires an new column, having all its cells fully reset to be used as subtrahends for all the other columns. Thereby, for every cell in an activated row, there is another unit, from the additional column that has I_{min} stored in it, making the reduction possible. For this case, the formulas are as follows:

$$x_{col_j} = f(I_{read}) = \frac{I_{read}}{(G_{max} - G_{min}) * V_{read}} * (2^N - 1) \tag{3.4}$$

$$x_{extra\ col} = f(k) = \frac{k * G_{min} * V_{read}}{(G_{max} - G_{min}) * V_{read}} * (2^N - 1) \tag{3.5}$$

$$x = x_{col_j} - x_{extra\ col} \tag{3.6}$$

It is important to note as well, that this second approach is automatically implemented in the differential mapping technique when it comes to interpreting. This data representation already requires double the columns to be implemented. Thu, no additional hardware is necessary for the interpretation task, as these are already in-built. Moreover, this mapping method does not need to only store 0s (HRSs) in every cell from every second column but any state, hence adding negative values to the total resolution quantity. This is made possible, thanks to the reduction of the two I_{min} from column j and j+1, remaining only to subtract the actual read current from the two, which makes the differential data representation to be from the start a compelling choice of memory mapping.

3.3 Power and Energy Calculation

Calculating and estimating the power and energy consumption of a crossbar array plays a decisive role in this study to evaluate different approaches properly. Various dependencies can influence the calculation, such as the WL voltage, the current flowing through the cell, the resistive state of the memristor, the input data and operations, and the length of different pulses. Besides determining values for the crossbar as a whole, the components of the cell are

also going to be individually analyzed. The power and energy calculation will be computed using a python script, with signals from the SPICE simulation, and the estimation will be done using different analytical formulas.

3.3.1 Low-Level Simulation

The SPICE simulation of the crossbar offers a detailed variety of available signals; hence a very accurate power and energy calculation is possible.

Approach

The implemented python script takes as input every WL voltage and BL current signal of the crossbar when calculating the total power of the cells by using the standard $P = V * I$ formula. Energy is given as the integral of power, which, in this case, can be presented as a Riemann sum,

$$E = \sum P * \Delta t \tag{3.7}$$

The power is multiplied by the step time between every new value written in the input file. Obviously, such integration might cause some calculation error if the step time does not tend to 0, but in this case, as it is extremely small, close to 0, it makes the calculation very precise.

The script calculates the energy consumption for the whole crossbar, but it can also do a more detailed computation for each input instruction. Therefore, besides the total crossbar consumption, there can be known how much each one of the read, write or mvm operations costs but also the total consumption for any particular cell.

Additionally, it might be interesting to know the energy costs of each component individually, as these can have different behaviors depending on the input operation. In order to do this, the voltage signal of the connecting point between the transistor and memristors from the SPICE simulation is needed. Furthermore, the power and energy formulas need to be slightly extended to:

$$P_{memristor} = (V_{WL} - V_{net}) * I_{cell} \quad and \quad P_{transistor} = V_{net} * I_{cell} \tag{3.8}$$

$$E_{memristor} = \sum P_{memristor} * \Delta t \quad and \quad E_{transistor} = \sum P_{transistor} * \Delta t \tag{3.9}$$

where V_{net} represents the connection point.

3.3.2 Energy Estimation

In this subsection, new analytical formulas will be presented to offer a satisfactory energy estimation compared to the more accurate approach using highly detailed data sets from the simulation. This analytical method is an estimation because the pulse format is idealized, as the rising and falling edges are considered to be purely vertical lines, compared to the simulation pulses, where the rise- and fall time of a pulse is **nonzero**. Furthermore, these formulas do not require knowledge of the current values but need the equivalent resistance or conductance of the cell for each unique operation. Additionally, the WL voltage and the input pulse length are necessary.

Write operation

The three studied data representations differ from one another only by how the mapping (write operation) is being performed. However, because the multibit and differential approaches work using the same principle of equally spaced conductance levels, only one energy formula can be utilized for these two, while the single-bit representation has one of its own. The general power formula for a single cell is now $P = \frac{V^2}{R}$.

For the multibit and differential data representations, the following energy equation has been designed to calculate the consumption of the set operation for a whole row based on its write input:

$$E_{row,set} = \sum_{i=0}^{2^N-1} k_i * (\frac{V_{set}^2}{R_{init_i}} * t_{set} + \frac{V_{set}^2}{2} * (\frac{1}{R_{final_i}} - \frac{1}{R_{init_i}}) * t_{set}) \tag{3.10}$$

k_i represents how many times value i occurs in the row, R_{init_i} the first measured set resistance of the pulse at i, R_{final_i} the last measured resistance at the end of the pulse, V_{set} the WL voltage, t_{set} the pulse length and N is the cell resolution. During the set pulses, the resistance has a slight linear increase, Figure 3.2, which requires a first-order approximation, done by the second part of this equation. It should be reminded that, to achieve lower energy consumption and because the set operation is always performed with the whole crossbar being already in HRS (only for the multibit and differential data representations), an input 0 during the write operation does not perform a new reset, which would be much more power-hungry, but a normal set with a gate voltage ($V_{gate} = V_{threshold}$) to reduce the energy costs and to keep the cell in the same HRS as desired. The write-reset formula looks then as follows:

$$E_{row,reset} = n * \frac{V_{reset}^2}{R_0} * t_{reset} \tag{3.11}$$

where n represents the number of columns in the crossbar. The total energy for a write-set or a write-reset operation is then equal to:

$$E_{total,set} = m * E_{row,set} \quad and \quad E_{total,reset} = m * E_{row,reset} \tag{3.12}$$

m being the number of used rows.

For the single bit representation, because there are only two states the reminded trick does not need to be applied, which results into having one single energy formula for the write operation.

$$E_{row,write} = k_0 * \frac{V_{reset}^2}{R_0} * t_{reset} + k_1 * (\frac{V_{set}^2}{R_{init_1}} * t_{set} + \frac{V_{set}^2}{2} * (\frac{1}{R_{final_1}} - \frac{1}{R_{init_1}}) * t_{set} \tag{3.13}$$

$$E_{total,write} = m * E_{row,write} \tag{3.14}$$

Read and MVM operation

For these two instructions, the same two energy equations can be used for all three approaches because, from the consumption perspective, only the cell resolution differs between them. The read formula is similar to the write, as it also goes through the cells from all the rows, one at a time.

$$E_{row,read} = \sum_{i=0}^{2^N-1} k_i * \frac{V_{read}^2}{R_i} * t_{read} \tag{3.15}$$

$$E_{total,read} = m * E_{row,read} \tag{3.16}$$

where $N = 1$ for the single-bit representation and $N \geq 2$ for the other two.

The MVM equation goes slightly deeper into the concept, as it needs to compute the result only for the activated rows, which means that the calculation of the MVM consumption happens directly for the whole crossbar.

$$E_{total,mvm} = \sum_{j=0}^{m} \alpha_j * (\sum_{i=0}^{2^N-1} k_i * \frac{V_{mvm}^2}{R_i} * t_{mvm}) \tag{3.17}$$

α_j representing the row activation factor at position j and taking only 1's (activated) or 0's (not activated) as values.

4 Results

This chapter will present an overview of the results obtained with the described framework, which will be divided into two categories, the MVM interpretation and the energy consumption. Each one will present general information from some specific designs, but it will also compare the results between the data representations using larger crossbar arrays. The experiments performed further will be based on a 32x32 input matrix, with a $N = 4$ bit cell resolution, with values ranging between -7 and 7. The 16th value, -8, is missing due to the incapacity of mapping it using the differential method, and it is desired that all the experiments are to be performed with identical input data. Moreover, the error for the MVM interpretation, as well as for the analytical energy consumption calculation will be estimated.

4.1 Data Preparation

Given that the input workload consists of a 32x32 matrix with a bigger resolution and the four designs used in these different experiments are single-bit, multibit with $N = 2$ and $N = 4$ as well as differential, $N = 3$, it is needed to perform bit-slicing or biasing to, first of all, correctly map the data in the crossbar. For the single-bit design, because this type of data representation cannot directly map negative numbers, all the values from the input matrix are to be incremented with a bias, in this case, 7, thus remaining only with positive numbers in the crossbar. Furthermore, these new values need to be sliced into four cells to be mapped, which leaves the actual crossbar as a 32x128-sized matrix. For the second design, multibit with 2-bit cell resolution, the same process is to be made, only that in this case, there are just two slices, resulting in a 32x64 crossbar. For the third case, multibit with $N = 4$, no slicing is needed as one cell can solely represent any number from the input, which has to be positive, thus, biasing is required, and a 32x32 crossbar is to be used. The final design does not require any particular setup, as the differential can map any value between -7 and 7, having a 3-bit cell resolution by doubling the columns, resulting in a 32x64-sized crossbar.

4.2 MVM Interpretation

The interpretation shall be performed as per the equations presented in Section 3.2.2. Additionally, interpreting tables, as Table 4.1 or Table 4.2 need to be made in order to know the conductance values for every state in which the cell finds itself, especially important being the values of G_{min} and G_{max}.

The accuracy of an MVM is dependent on two significant aspects. The first one relies on how accurately the mapping was performed, which is given by the IGVVA. Tiny gate increases, even smaller than 10mV, can have a considerable impact on the mapping performance, especially for the first conductance levels, where the gate voltages are quite near to each other,

Value	State	Current	Equivalent conductance	Equivalent Resistance
0	HRS0	2.8711 μA	9.57 μS	104.49 kΩ
1	LRS1	10.867 μA	36.223 μS	27.606 kΩ
2	LRS2	18.853 μA	62.843 μS	15.912 kΩ
3	LRS3	26.845 μA	89.483 μS	11.175 kΩ

Table 4.1: 1T1R cell states for multibit, 2 bit resolution, using a 32-HP transistor

Value	State	Current	Equivalent conductance	Equivalent Resistance
0	HRS0	3.0268 μA	10.089 μS	99.114 kΩ
1	LRS1	22.291 μA	74.303 μS	13.458 kΩ
2	LRS2	41.555 μA	138.52 μS	7.2193 kΩ
3	LRS3	60.819 μA	202.73 μS	4.9326 kΩ

Table 4.2: 1T1R cell states for multibit, 2 bit cell resolution, using a 130nm transistor

as it can be seen in Figure 3.5. Clearly, increasing the cell resolution makes the first gate voltages even tighter to one another, which requires even better precision for programming. The second accuracy aspect consists of how many rows there are activated in a crossbar. The more rows, the more values on the same column to be added and the bigger the error.

4.2.1 Output Evaluation between Data Representations

After reverting the biasing and the bit-slicing where was needed, the following interpretation errors were measured using the above-specified input matrix and having all the rows activated for the MVM operation:

Design	Cell resolution	Representation	Avg. Error	Actual crossbar size
A, single bit	1	bias & slicing	0.295%	32x129
B, multibit	2	bias & slicing	4.564%	32x65
C, multibit	4	bias	6.142%	32x33
D, differential	3	-	0.755%	32x64

Table 4.3: Designs' table, 32nm transistor

The average MVM errors from Table 4.4 would be even smaller in case of design A and B, but due to the bit-slicing, the errors are being scaled exponentially with the number of slices. In order to be able to correctly interpret the result of a MVM, the inaccuracy cannot be bigger than 50%, otherwise, the interpreter could not know how to properly detect the result. Case C has the highest average error due to the many levels needed to be mapped in a single cell, which creates smaller buffers between each other and makes precise programming more difficult. The last design shows a promising average error, although the cell resolution is 3. It is also worth noting, that in the last column of Table 4.4 the crossbar size is increased by one column compared to what was specified previously for the first three cases. This is made for

Design	Cell resolution	Representation	Abs. avg. error	Actual crossbar size
A, single bit	1	bias & slicing	0.002	32x129
B, multibit	2	bias & slicing	0.019	32x65
C, multibit	4	bias	0.031	32x33
D, differential	3	-	0.006	32x64

Table 4.4: Designs' table, 32nm transistor

Design	Cell resolution	Representation	Abs. avg. error	Actual crossbar size
A, single bit	1	bias & slicing	0.002	32x129
B, multibit	2	bias & slicing	0.019	32x65
C, multibit	4	bias	0.031	32x33
D, differential	3	-	0.006	32x64

Table 4.5: Designs' table, 32nm transistor

the practical application, where the interpreting process needs to subtract G_{min} from every cell for an accurate MVM result. Therefore, adding an additional column that contains fully reset cells is mandatory. Only case D does not need this extra step, as it is already present in its implementation.

Figure 4.1 shows another example of MVM when random rows from the crossbar are activated. The single-bit representation keeps a tiny inaccuracy in this case as well, with the differential approach coming second and having around 0.55% average error. Design B has error values between 1% and 4%, averaging at around 2%, while design C has once again the highest average error, 3.28%, with a maximum error value of up to 7%.

4.3 Power and Energy Consumption

The results in terms of energy consumption will be presented in this section. Firstly, particular cells will be examined, as the costs of every unique operation will be calculated for a generic example. These costs will be divided in total, memristor and transistor usage. Secondly, the above-mentioned experiments and designs will also be tested to gather results about the total energy consumption of a bigger crossbar, which will happen for cells with two different transistor models. At last, a comparison between the simulated results and the analytical version will be performed.

4.3.1 Cost of Unique Operations

The following example contains only one cell, a 1x1 crossbar, with a 2-bit cell resolution, and all the possible operations are performed. The order of the instructions is set, read and reset for all four states. Figure 4.2 shows the plot of power over time, where the blue line represents the total power, the orange curve the memristor, and the green the power consumption of the transistor.

Table 4.5 presents all possible combinations of the total energy usage in a 2-bit resolution

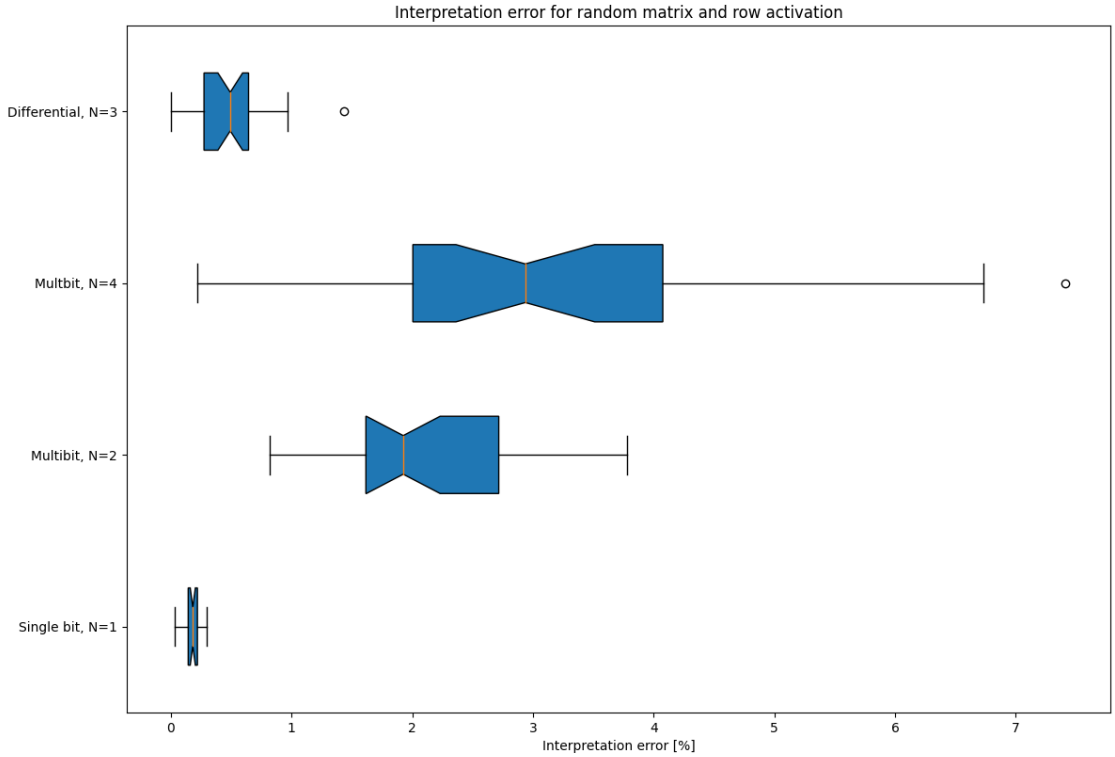


Figure 4.1: Box and whisker plot for the interpretation error

cell for any given instruction. Moreover, it shows how the energy is distributed on each component individually. The reset operation is by far the most expensive, as it, first of all, has a much longer time pulse. In all the states, the memristor uses most of the energy for this operation. For the set instructions, with the increase of gate voltage, it can be observed that the transistor, from an almost even distribution, starts draining much more power. The consumption of the read operation is dominated by the memristor in the more resistive states, followed by a much higher consumption from the transistor in the less resistive states.

4.3.2 Power Comparison between Data Representations

The results of the four mentioned designs can be found in Figure 4.3 and Figure 4.4 as a comparison between the two transistor models, where the blue colored columns represent the configuration with the 32nm transistor and the orange columns the 130nm one. The three data representations discussed previously were compared in terms of energy consumption for the write, respectively, for the mvm operation, with the multibit representation having two scenarios of cell resolution. Starting with the MVM results, it can be observed that the single-bit is the least efficient method, consuming almost twice as much as multibit with 2-bit cell resolution and four times more than multibit with $N = 4$. The differential approach with a cell resolution of three uses 20% more energy than design C for the mvm operation in the case of the 130nm transistor, while for the smaller model, it consumes 25% more. Overall, for both 1T1R cells, the multibit with $N = 4$ provides the more efficient consumption, followed shortly by the differential.

For the write operation, a similar outcome is shown, but with a change in energy consumption between the two transistor models. In the MVM case, the cell with the bigger transistor had a much higher power usage due to the fact that the read currents for every state were much higher than the ones from the 32nm model. Here, the smaller transistor is the less efficient

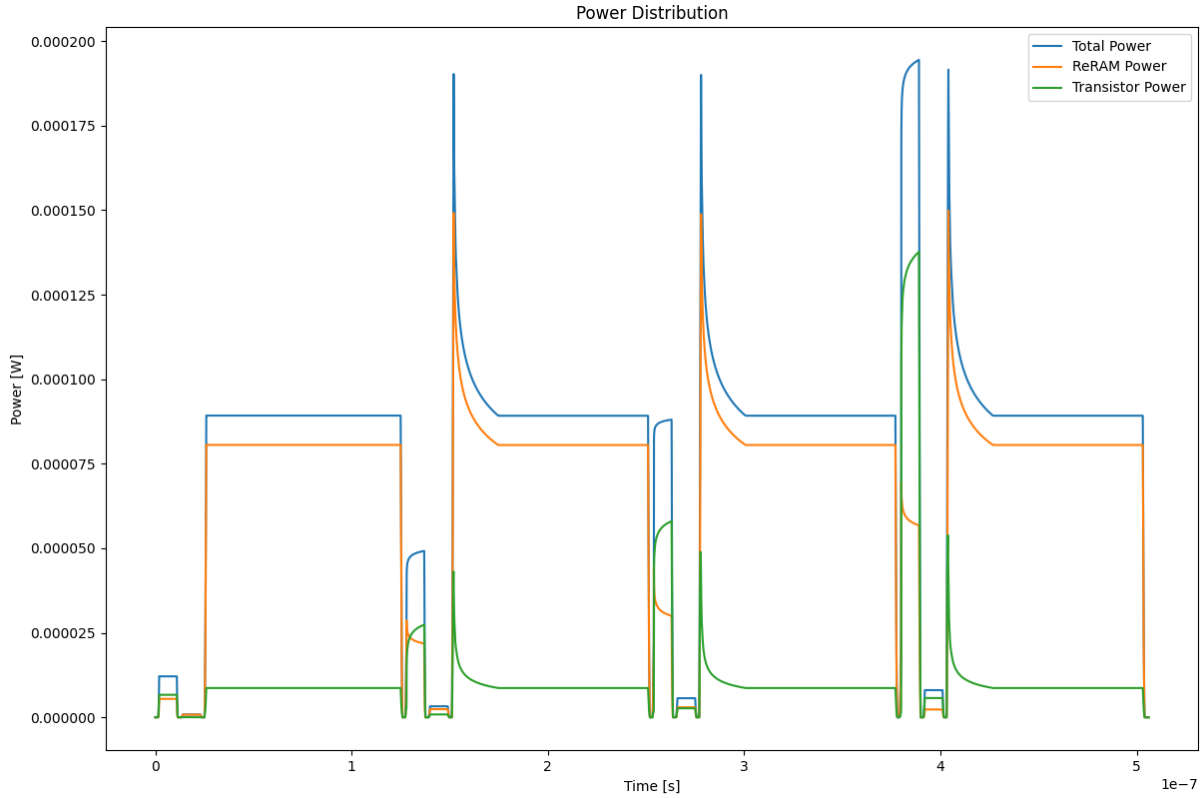


Figure 4.2: Plot of the power consumed by instructions

Operation	State	Total Energy	Memristor percentage	Transistor percentage
Read	HRS0	8.27 fJ	92.38%	7.62%
Set	HRS0	0.113 pJ	47.8%	52.2%
Reset	HRS0	8.895 pJ	90.1%	9.9%
Read	LRS1	31.28 fJ	70.72%	29.28%
Set	LRS1	0.451 pJ	43.36%	56.64%
Reset	LRS1	9.253 pJ	87.40%	12.60%
Read	LRS2	51.42 fJ	48.18%	51.82%
Set	LRS2	0.821 pJ	35.8%	64.2%
Reset	LRS2	9.268 pJ	86.65%	13.35%
Read	LRS3	76.84 fJ	17.81%	82.19%
Set	LRS3	1.802 pJ	28.17%	71.83%
Reset	LRS3	9.274 pJ	83.79%	16.21%

Table 4.6: List of values for each operation, 2 bit resolution, 32nm HP transistor

one. As in the previous case, the multibit method with a cell resolution of four is by far the least energy-draining, being at least twice as efficient as any other representation examined for both models. Looking at the 32nm transistor, the differential method is 17.4% more energy efficient than the single-bit, while the multibit, $N = 2$, with 12.7%. For the 130nm model, the efficiency is higher when switching from the single-bit data representation to the

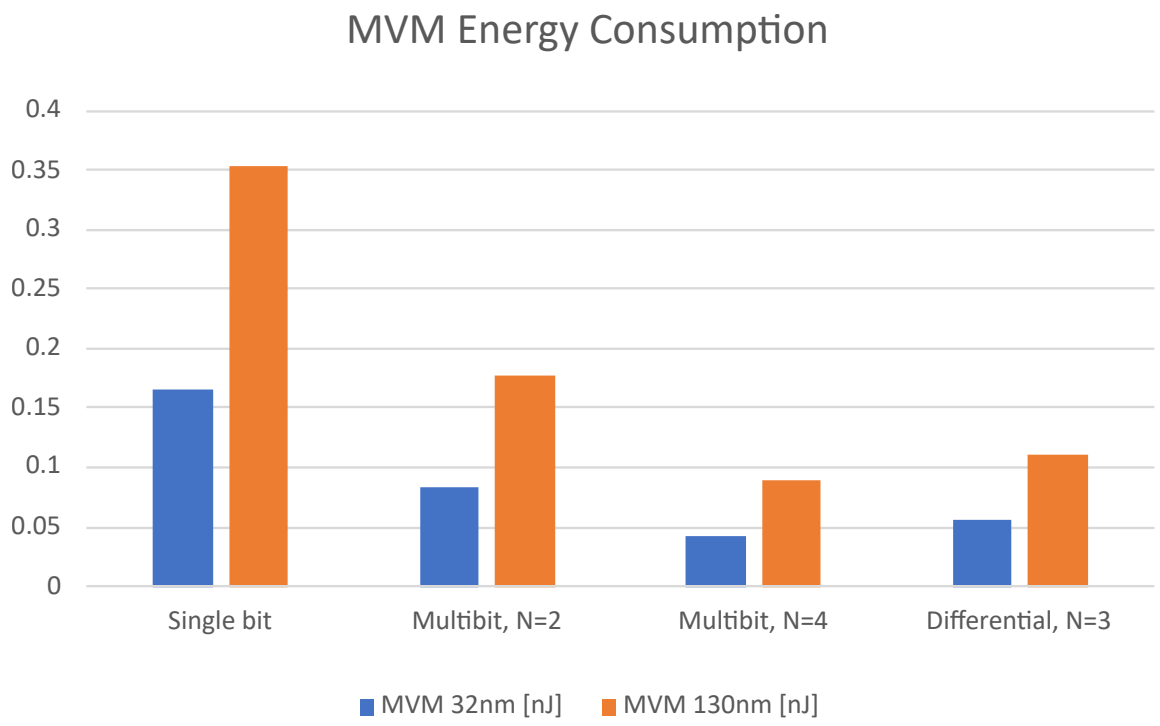


Figure 4.3: Energy consumption for the MVM operation

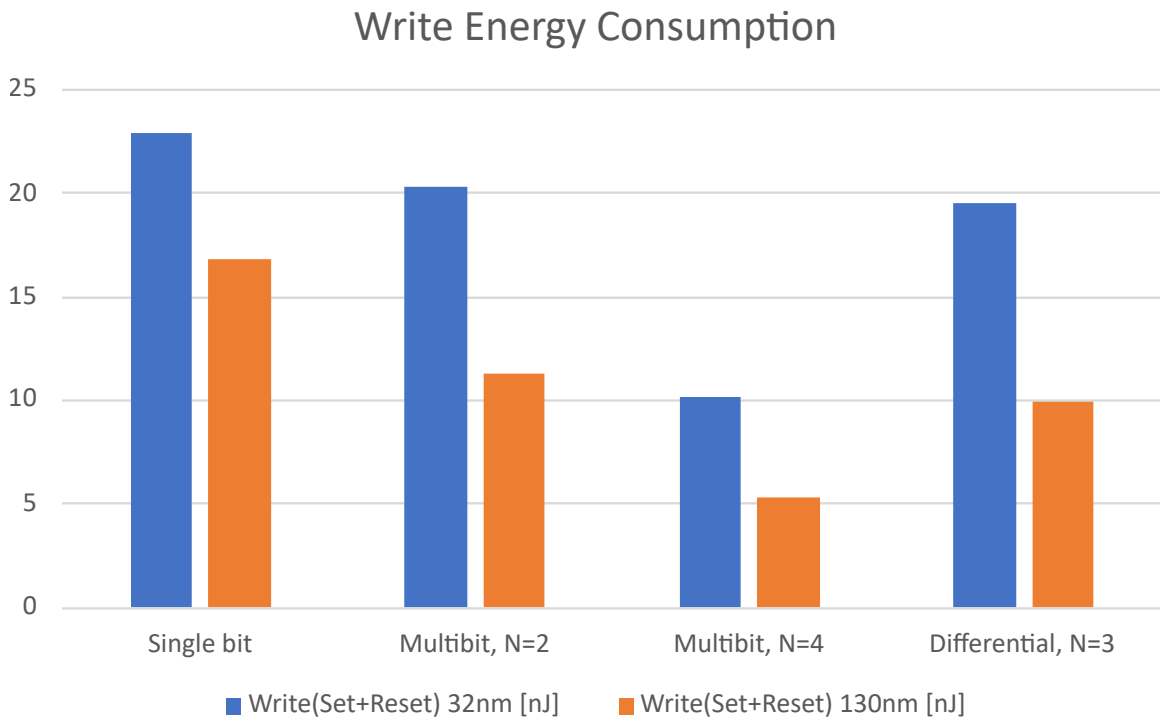


Figure 4.4: Energy consumption write operation

other two, as the multibit, $N = 2$ has a 33% decrease in energy used and the differential 40.8% improvement over it. As mentioned above, multibit with $N = 4$ produces an efficiency

of 68.6%, which means it consumes 3.18 times less energy than the single-bit representation.

It should also be mentioned that for all three data representations, the reset operation consumed by far the most energy. This affects the multibit and differential representations considerably, as after the mvm is performed, the cells need to be reset to be able to store new data, which means a total reset for the whole crossbar is needed. Using an even shorter reset pulse, 25ns, which was the lowest limit found for this operation to work correctly, would also further increase the performance of the two mentioned mapping techniques over the single-bit representation.

The crossbar size represents another essential aspect when looking at energy efficiency. Mapping with the multibit and differential approaches gives the possibility of achieving better energy performances thanks to needing to use smaller crossbars. The ability to store more values in a single cell eliminates the need to always use bit-slicing to represent bigger numbers. The multibit design with $N = 4$ uses, as mentioned above, a 32x32 crossbar, exactly the size of the input matrix. It achieves much better energy efficiency than the single-bit, which had to use four times as many cells to represent the same matrix.

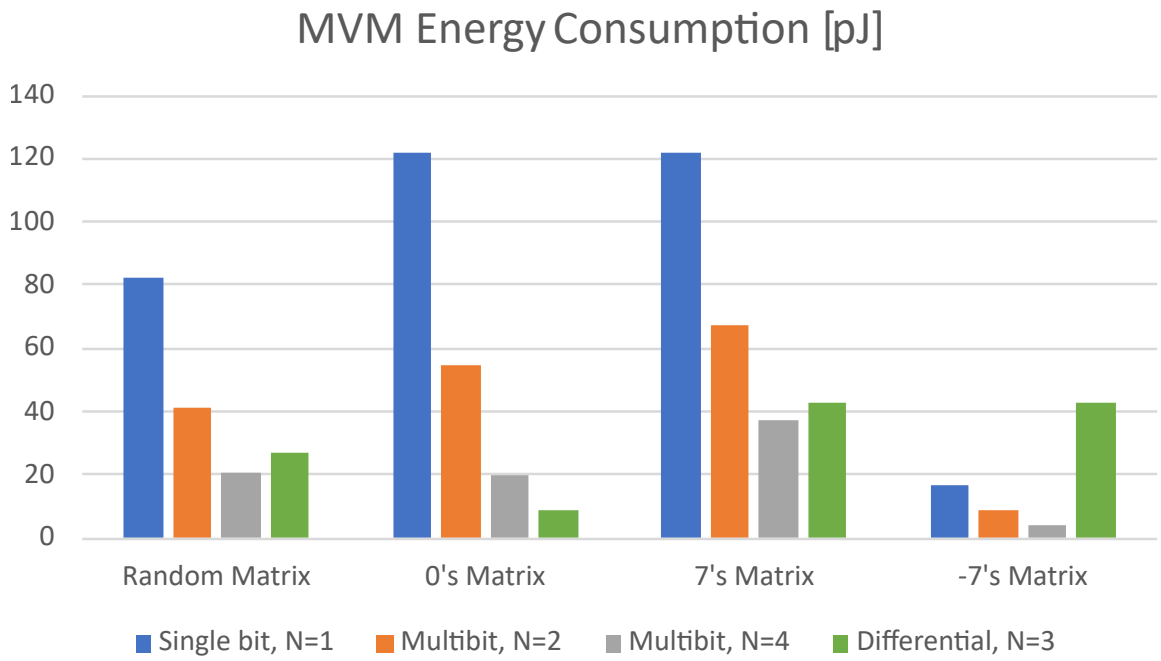


Figure 4.5: MVM energy consumption for 4 experiments with different matrices and random row activation, 32nm transistor

Figure 4.5 presents the results of three additional test matrices next to the above-described random matrix. These new examples contain for each case only 0s, 7s, and -7s, which are the more interesting values of the interval. The row activation for the MVM operation is performed randomly for this simulation, which significantly reduces the consumption for the random matrix, as only around half of the rows are activated. When the whole input is 0, which is generally an often used value in MVMs, the differential data representation has a much lower consumption than the other three, with the multibit, $N = 4$, being close to it, but still having around three times bigger consumption. It should be reminded that value 0 is seen as a 0 + bias for these three designs; therefore the consumption is higher for them.

When looking at the 7s matrix, which is the highest possible mapped value in this scenario, it can be observed that design C slightly outperforms design D. Mapping a matrix of -7s actually means mapping only the value 0 in the case of the first three designs, due to the addition of the bias. This explains why all these three techniques are far more efficient than the differential one, which has to store 0s and 7s to obtain -7s.

4.3.3 Accuracy of Estimated Energy

This section will examine the error between the calculated energy values using the low-level simulation traces and an estimation using analytical formulas. Additionally, interpreting tables, like Table 4.1 but for all the operations, need to be provided to be able to use the formulas accordingly.

The operations of the above-mentioned 1x1 crossbar are estimated and compared to the total calculated energy. First of all, the error is induced, as mentioned in Chapter 3, due to the rising and falling edges of the pulses, which are not instant but take 1ns to be performed. Secondly, as per Figure 4.2, the write signals show a slight non-constant behavior, which increases the estimated error. The set operations, excluding the HRS0 case, always show a logarithmic increase, while the reset presents very short and abrupt spikes right at the beginning. Thus, the equivalent resistances or conductances which are only constants in the interpreting tables will induce slight errors.

In Table 4.6, it can be observed that state HRS0 has, for each operation, the lowest error between all four states, this being due to the initialized resting state of the memristor. On average, the read operation has a 5.425% analytical error, the set a 5.287%, and the reset a 2.84% for this two-bit cell resolution case.

Operation	State	Calculated Energy	Estimated energy	Error
Read	HRS0	8.27 fJ	8.613 fJ	3.98%
Set	HRS0	0.113 pJ	0.121 pJ	6.61%
Reset	HRS0	8.895 pJ	8.925 pJ	0.34%
Read	LRS1	31.28 fJ	32.60 fJ	4.05%
Set	LRS1	0.451 pJ	0.472 pJ	4.45%
Reset	LRS1	9.253 pJ	8.932 pJ	3.59%
Read	LRS2	51.42 fJ	56.56 fJ	9.08%
Set	LRS2	0.821 pJ	0.860 pJ	4.53%
Reset	LRS2	9.268 pJ	8.937 pJ	3.70%
Read	LRS3	76.84 fJ	80.54 fJ	4.59%
Set	LRS3	1.802 pJ	1.908 pJ	5.56%
Reset	LRS3	9.274 pJ	8.941 pJ	3.72%

Table 4.7: Analytical error, 2 bit resolution, 32nm HP transistor

Estimating the energy consumption for design B, multibit with a cell resolution of two, using table Table 4.6 results in the following Table 4.10. It can be noticed that the estimation error

of this design is quite close to the average errors calculated above.

Operation	Calculated Energy	Estimated Energy	Error
MVM	0.083 nJ	0.087 nJ	4.59%
Set	1.506 nJ	1.588 nJ	5.16%
Reset	18.786 nJ	18.295 nJ	2.68%

Table 4.8: Estimated Energy and its error for design B with full row activation for the MVM and 32nm HP transistor

Operation	Calculated Energy	Estimated Energy	Error
MVM	0.055 nJ	0.057 nJ	3.50%
Set	0.935 nJ	0.986 nJ	5.48%
Reset	18.54 nJ	18.19 nJ	1.92%

Table 4.9: Estimated Energy and its error for design B with full row activation for the MVM and 32nm HP transistor

Operation	Calculated Energy	Estimated Energy	Error
MVM	0.0413 nJ	0.0431 nJ	4.18%
Set	0.676 nJ	0.713 nJ	5.19%
Reset	9.463 nJ	9.139 nJ	3.54%

Table 4.10: Estimated Energy and its error for design B with full row activation for the MVM and 32nm HP transistor

Operation	Calculated Energy	Estimated Energy	Error
MVM	0.165 nJ	0.172 nJ	4.07%
Set	3.703 nJ	3.911 nJ	5.30%
Reset	19.959 nJ	19.492 nJ	2.41%

Table 4.11: Estimated Energy and its error for design B with full row activation for the MVM and 32nm HP transistor

4.4 Overall Comparison between Data Representations

Figure 4.6 presets a final analysis between the studied mapping techniques looking at area usage, energy consumption and MVM error interpretation. The columns are normalized, between 1 and 0, against the highest value in each category, to offer a better view on the results. This test used the above-specified random matrix and randomly activated rows. It

can be observed, in terms of area usage and MVM energy consumption, that the single-bit offers the worst results, but excels at the interpretation accuracy. Design B shows a balanced outcome with a good improvement in the first two metrics over the single-bit technique, but has a higher interpretation error than it. The multibit representation with a cell resolution of four offers the smallest achievable crossbar size within this scenario, with a significant improvement in efficiency as well, but has the lowest interpretation accuracy among all the designs. The jack-of-all-trades of this thesis, the differential approach, shows promising results in all three metrics. Its used surface is twice as big as the design's C area and has a slightly increased energy consumption than it as well, but in return, the interpretation error is much smaller.

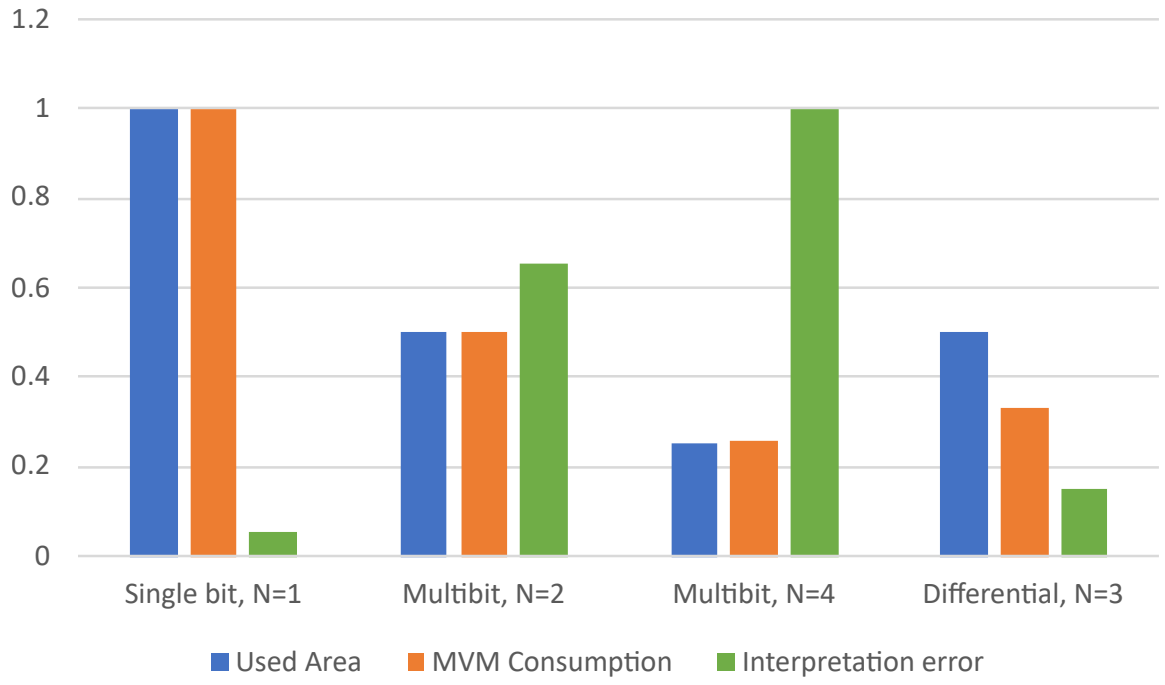


Figure 4.6: Normalized comparison of area, energy consumption and error interpretation, using the 32nm transistor

5 Conclusion

5.1 Summary of the Thesis

This thesis aimed to evaluate different data representations for memristor-based matrix-vector multiplication engines. Firstly, a tool, which allows user input for the crossbar size, operations, and parameters, was built to generate netlist files to be transient simulated with the Spectre Simulator offered by Cadence. This tool needed to contain, besides the classic, single-bit, programming, also the multibit mapping technique, which made storing up to 2^N values inside of a cell possible, using the IGVVA. The differential data representation was therefore built based on the multibit programming, but allowed to directly store negative values as well, by the cost of doubling the columns, without having to add a bias, as needed in the case of the other two when mapping negative numbers. Secondly, after accessing the results of the simulation, another script was implemented to interpret the results of the MVM process and to estimate their error. For this step, proper interpreting formulas had to be designed for both cases of equivalent conductance levels. Besides the sensed current in the column after the MVM, the row voltage, as well as the minimum and the maximum conductance of the unit need to be known to correctly interpret the results. The impact of G_{min} needed to be subtracted from what is being sensed because in both the cases of single and multibit mapping the value 0, HRS0, did not result in zero measured current, but in $G_{min} * V_{read}$. Only the differential method did this implicitly, due to its feature to subtract the current of every two paired columns, in this way, not only removing the G_{min} error but also allowing direct negative programming. At last, the energy consumption was measured in two ways, by calculating using the signals directly from the simulation and by using analytical formulas to estimate this consumption.

As a conclusion of the obtained results, after evaluating different data representations, the following assumptions can be made. All three studied data representations have their advantages and disadvantages, but in terms of efficiency and area usage, both the multibit and the differential approaches revealed a significant improvement over the single-bit representation. As the results in the previous chapter showed, the bigger the cell resolution N was becoming, the higher the efficiency and lower the required number of cells. Choosing the appropriate N is an essential step, as too big resolutions become harder to be mapped correctly, as they require very precise gate voltages. Looking at the interpretation accuracy, the single-bit had the best results, followed really close by the differential approach, which also had very accurate interpretations, while the error of the multibit representation was about five times bigger than that of the differential. Considering all the aspects, the data representation with the most benefits and that provided overall great results was the differential mapping technique.

5.2 Future Work

Addition of wire resistance and parasitic capacitance effects

As the 1T1R crossbar has a compact structure, parasitic effects and wire resistance are existent in the device. These could be added to the existing framework for a more realistic power and energy consumption calculation.

Memristor variability

Since the memristor is not an ideal component, it is then subject to variability. Therefore, the inclusion of device-to-device and cycle-to-cycle variability would offer a better prospect for real-world scenarios. The JART model employed in this thesis, offers parameters to include both these types of variability in the simulations.

Voltage sensing differential representation

An improvement over the previously described differential technique, which functions based on a current sensing mechanism, would be switching to a voltage-mode sensing scheme. [16] shows an increase in efficiency of about 3.5 times higher than the current measuring method, while performing MVM operations.

Energy calculation for the output peripherals

Another interesting aspect for future work is calculating the energy consumption of the sensing peripheral circuits, such as analog to digital converters or shift and add components.

List of Figures

List of Tables

List of Listings

References

[1] A. Shafiee, A. Nag, N. Muralimanohar, R. Balasubramonian, J. P. Strachan, M. Hu, R. S. Williams, and V. Srikumar, "Isaac: A convolutional neural network accelerator with in-situ analog arithmetic in crossbars," *ACM SIGARCH Computer Architecture News*, vol. 44, no. 3, pp. 14–26, 2016.

[2] A. Huang, X. Zhang, R. Li, and Y. Chi, "Memristor neural network design," *Memristor and Memristive Neural Networks*, pp. 1–35, 2018.

[3] M. Zangeneh and A. Joshi, "Design and optimization of nonvolatile multibit 1t1r resistive ram," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 8, pp. 1815–1828, 2013.

[4] C. Bengel, D. K. Zhang, R. Waser, and S. Menzel, "Jart vcm v1 verilog-a compact model."

[5] C. Denker, "Power modeling of memristor-based circuits," *Bachelor's thesis, Institute for Communication Technologies and Embedded Systems (ICE), RWTH Aachen University*, 2022.

[6] J. Wen, M. Ulbricht, E. Perez, X. Fan, and M. Krstic, "Behavioral model of dot-product engine implemented with 1t1r memristor crossbar including assessment," in *2021 24th International Symposium on Design and Diagnostics of Electronic Circuits & Systems (DDECS)*. IEEE, 2021, pp. 29–32.

[7] V. Milo, F. Anzalone, C. Zambelli, E. Pérez, M. K. Mahadevaiah, Ó. G. Ossorio, P. Olivo, C. Wenger, and D. Ielmini, "Optimized programming algorithms for multilevel rram in hardware neural networks," in *2021 IEEE International Reliability Physics Symposium (IRPS)*. IEEE, 2021, pp. 1–6.

[8] V. Milo, C. Zambelli, P. Olivo, E. Pérez, M. K. Mahadevaiah, O. G. Ossorio, C. Wenger, and D. Ielmini, "Multilevel hfo2-based rram devices for low-power neuromorphic networks," *APL Materials*, vol. 7, no. 8, p. 081120, 2019.

[9] G. Pedretti and D. Ielmini, "In-memory computing with resistive memory circuits: Status and outlook," *Electronics*, vol. 10, no. 9, p. 1063, 2021.

[10] W. Wan, R. Kubendran, C. Schaefer, S. B. Eryilmaz, W. Zhang, D. Wu, S. Deiss, P. Raina, H. Qian, B. Gao *et al.*, "Edge ai without compromise: Efficient, versatile and accurate neurocomputing in resistive random-access memory," *arXiv preprint arXiv:2108.07879*, 2021.

[11] M. Le Gallo, S. Nandakumar, L. Ciric, I. Boybat, R. Khaddam-Aljameh, C. Mackin, and A. Sebastian, "Precision of bit slicing with in-memory computing based on analog phase-change memory crossbars," *Neuromorphic Computing and Engineering*, vol. 2, no. 1, p. 014009, 2022.

[12] R. Pelke, "Evaluation of modeling techniques for cycle-accurate simulation of neuromorphic systems," *Master's thesis, Institute for Communication Technologies and Embedded Systems (ICE), RWTH Aachen University*, 2022.

[13] M. Hu, J. P. Strachan, Z. Li, E. M. Grafals, N. Davila, C. Graves, S. Lam, N. Ge, J. J. Yang, and R. S. Williams, "Dot-product engine for neuromorphic computing: Programming 1t1m crossbar to accelerate matrix-vector multiplication," in *2016 53nd acm/edac/ieee design automation conference (dac)*. IEEE, 2016, pp. 1–6.

[14] P. Packan, S. Akbar, M. Armstrong, D. Bergstrom, M. Brazier, H. Deshpande, K. Dev, G. Ding, T. Ghani, O. Golonzka *et al.*, "High performance 32nm logic technology featuring 2 nd generation high-k+ metal gate transistors," in *2009 IEEE international electron devices meeting (IEDM)*. IEEE, 2009, pp. 1–4.

[15] D. Arbet, V. Stopjaková, M. Kováč, L. Nagy, M. Rakús, and M. Šovčík, "130 nm cmos bulk-driven variable gain amplifier for low-voltage applications," *Journal of Circuits, Systems and Computers*, vol. 26, no. 08, p. 1740003, 2017.

[16] W. Wan, R. Kubendran, B. Gao, S. Joshi, P. Raina, H. Wu, G. Cauwenberghs, and H. P. Wong, "A voltage-mode sensing scheme with differential-row weight mapping for energy-efficient rram-based in-memory computing," in *2020 IEEE Symposium on VLSI Technology*. IEEE, 2020, pp. 1–2.