POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Design of a benchmarking platform for Logic-In-Memory architectures based on ferroelectric HfO_2

Relatore:
prof. Mariagrazia Graziano

Candidato:
Luca Mozzone

Aprile 2021

Summary

Silicon devices have undergone a massive down scaling process in the last decades, this brought transistors near the physical limitations introduced by quantum mechanics. In this area the boundaries of conventional computation are stretched to their limits in order to increase performances and decrease power consumption. This path is rapidly becoming more expensive and major breakthroughs are more difficult to reach. In this scenario different technologies are being investigated to fill the holes left by silicon, especially computing technologies that can be non-volatile.
Hafnium dioxide (HfO_2) stands out in this landscape because of its ferroelectric properties that make of it an intrinsic memory. Moreover, its use as high K insulator in modern CMOS processes makes it perfect for integration with existing devices and production steps.
At the same time evolution in computing architectures has been minimal with respect to the one of the hardware: modern systems still implement fined tuned versions of the Von Neumann architectures. This introduced a mismatch in performances, mainly created by the huge amount of time and energy spent on communications between the memory and the processor unit, the so called Von Neumann bottleneck. The task to explore the design space is impressive given the numerous degree of freedom introduced by both device and system level. This thesis proposes a benchmarking platform designed in SYSTEMC. Its aim is to speed up the process by simulating and then comparing different solutions both in the technological and in the architectural department. At the same time the estimation must remain as faithful as possible to the nature of different implementations.
Furthermore, the platform could be employed to find strengths and weaknesses of the various emerging technologies with respect to each other and CMOS. This could initiate a feedback loop with device level research to help to improve the technological nodes and by reflection the overall performances.
This thesis is divided in three chapters. The first one is devoted to the ferroelectric properties of HfO_2, since it is the main target technology for this stage of the project. The physical phenomenon is introduced, the role of this material in CMOS processes is underlined and an overview of the main ferroelectric devices is given.

The second chapter provides an exhaustive taxonomy of the ever-growing Logic-in-Memory (LiM) architectures field. The most important paradigms are explained and compared with each other.
The last chapter focuses on the realization of the simulation platform. The design of the structure is unraveled, the different components are analyzed together with how they interact.
The current state of the platform allows for LiM simulations in both CMOS and ferroelectric technology with different architectures, but the quantitative results are still incomplete due to the platform not being mature enough. However, they prove how this tool has the potentialities to rise up to the challenge.

Contents

Chapter 1

Technology

Ferroelectricity is the property of some materials of having an intrinsic electric polarization. This effect can be exploited to create new kind of memory devices and variable capacitors, because of this reason the research on them is grown in the last decades. This chapter, after a general introduction of the phoenomena, will be focused on HfO_2 ferroelectric thin films due to their great compatibility with the CMOS technology, concluding the chapter with an overview of the main devices developed exploiting this effect.

1.1 Physical phenomenon

The term ferroelectric determines a material which, in absence of external electric field, exhibits a spontaneous electric polarization. The name is derived by the analogy with ferromagnetism, when it's been observed for the first time by Valasek in Rochelle salt in 1920 [1], where the presence of an intrinsic magnetic moment is the key characteristic.

When an electric field (E) is applied to a material it exhibits a polarization density (P) that can be different based on its properties. The majority of materials have a linear relationship between the two quantities, as shown in Fig 1.1, they are called dielectrics. Paraelectric materials have instead a nonlinear correlation in which the polarization slope is not constant with the electric field, their characteristic response can be seen in Fig. 1.2.

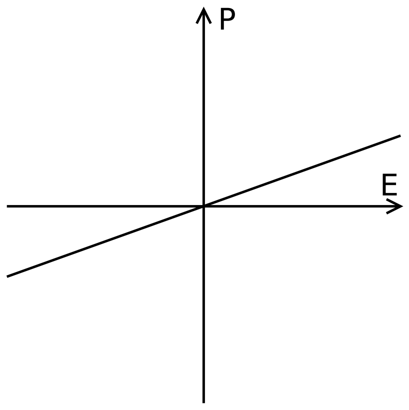

Figure 1.1. Dielectric polarization response.
From `https://tinyurl.com/yxrj3l55`

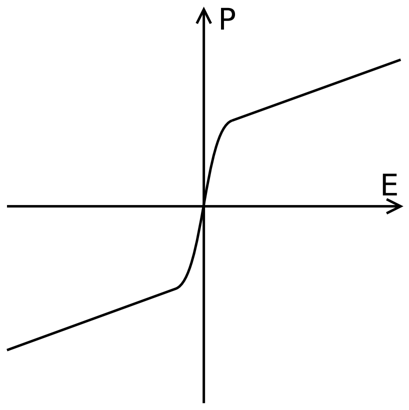

Figure 1.2. Paraelectric polarization response.
From `https://tinyurl.com/y5tasxx8`

Ferroelectric ones differs from the latter because of the presence of a remaining polarization, it can be reversed if exposed to a suitable electric field. This means that a ferroelectric response is not only influenced by the applied field but also by the current state of the polarization, this dependency on the material history gives rise to an hysteresis loop (Fig. 1.3).

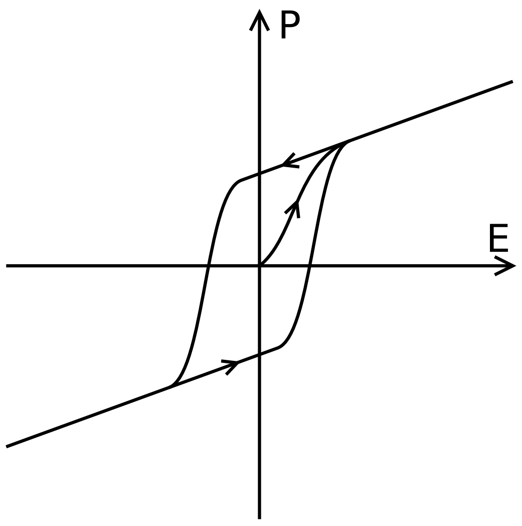

Figure 1.3. Ferroelectric polarization response. From `https://en.wikipedia.org/wiki/File:Ferroelectric_polarisation.svg`

When the electric field is not applied the polarization can be defined as:

$$\vec{P} = \sum \frac{\vec{p}}{V} \tag{1.1}$$

where $\vec{p}$ is the dipole moment of the single molecule and V is the total volume. If the atoms are considered point charges Q_i the dipole moment can be further expanded into:

$$\vec{p} = \sum_i Q_i \vec{R}_i \tag{1.2}$$

in which the $\vec{R}_i$ is the i-th atom position.
From this representation we can see that the presence of asymmetry in the material's basic cell is a key component for ferroelectricity to be exhibited. In fact if the crystal was symmetric the various opposite position vectors would cancel each other out resulting in a null dipole.

The ferroelectric crystal structure produces in this way preferential axis along which the various dipoles align resulting in a lower energy state and therefore they are preferred by the system. Taking as an example a one dimensional crystal, composed by two kinds of atoms with opposite charge, we can foresee two stable states: one in which the dipoles are all aligned towards the left, and one in which they are aligned towards the right; since they are equivalent they have the same resulting energy. The configurations of the system that places in between these two have higher energy, resulting in a potential barrier to be overcome in order to switch between the two.

We can induce a state change applying an electric field and changing therefore the electric polarization energy $E_p = -\vec{P} \cdot \vec{E}$ breaking the balance between the two energy states. If the electric field is strong enough to overcome the barrier the crystal switches state into the most stable one. This is the basic principle that gives rise to the hysteresis loop as shown in Fig. 1.4. In fact after the electric field is removed the energy local minimums return to be equal again but the polarization is now in the opposite direction.

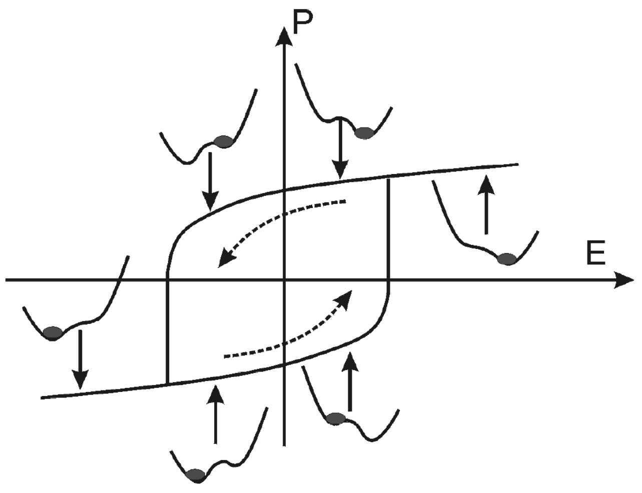

Figure 1.4. Visual representation of the hysteresis loop. When an electric field is applied the energy level are tilted until the barrier is overcome switching the system' state. From Litlewood, 2002 [2].

If we instead consider a thin film of ferroelectric material it can be seen as a 2D system. In these circumstances the polarization axis can be parallel to the film plane or orthogonal to it; the former case is referred to as a in-plain polarization while the latter as out-of-plane. In order to induce a state switch in a 2D ferroelectric system also the orientation of the electric field must be correct, if this is orthogonal to the polarization axis it won't be able to influence the system.

The various cells organize themselves into ordered structures inside the material. The directions of their axis depend on the electrostatic and mechanical conditions of the crystal. The region in which the spontaneous polarization axis are aligned is defined as ferroelectric domain, the regions between two different domains are called domain walls [3].
During phase transitions a surface charge accumulates creating a depolarizing electric field E_d oriented in the opposite direction with respect to the spontaneous polarization, its intensity can reach a level for which a single domain is no more energetically preferable. If that happens the system arranges itself into different domains with opposite P in order to minimize electrostatic energy. When a mechanical

stress is present the axis of the basic cells tend to point orthogonally to it, while in the areas not affected by it the axis remains parallel, minimizing in this way elastic energy.

In the end the material is composed by different domains with different preferred directions for polarization, this influences drastically the global polarization since it is no more a two state system but given the high number of domains that usually are formed into a macroscopic crystal it's more similar to a continuous one. The axis into an as-grown crystal can be in random order resulting in a negligible net polarization. The various dipole can be oriented through an electric field as shown in Fig. 1.5.

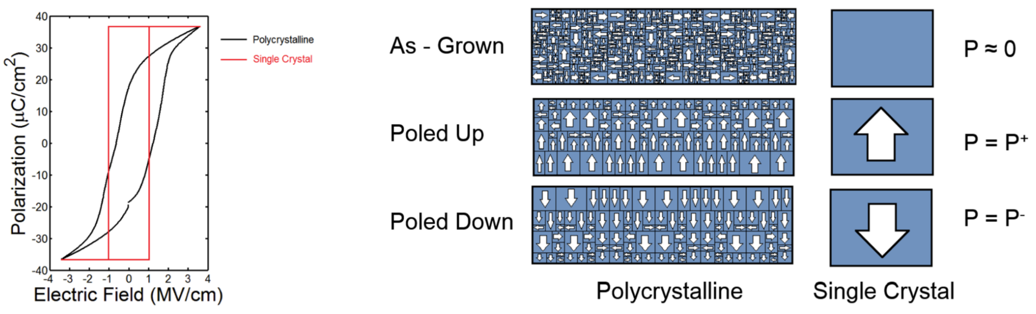

Figure 1.5. On the left single crystal square polarization vs. a smooth polycrystalline one. On the right a representation of aligned domains through an external electric field and resulting macroscopic polarization. From Lomenzo, 2016 [4].

1.2 Ferroelectricity in HfO_2 thin films

Ferroelectricity is a property typical of those materials that present a non-centrosymmetrical crystal structure. This produces a charge asymmetry in the basic cell, resulting in an electric field. In particular bulk hafnium dioxide is usually in its monoclinic phase, it is stable at room conditions and it's centrosymmetrical thus does not present ferroelectric behaviour.
In thin film form the different influence of surface energy makes more stable a different phase of HfO_2, the tetragonal one, usually not present in considerable quantities in bulk samples. The latter is still symmetrical, so it is still a dielectric phase.
It has been observed that HfO_2 thin films, deposited through atomic layer deposition, nucleates in a tetragonal phase changing eventually to a monoclinic one during cooling through a process that involves an expansion and the shearing of the basic cell [5]. If the process is mechanical constrained through TiN electrodes, deposited by chemical vapor deposition, they prevent the physical deformation of the cells and allow the formation of a different phase: the orthorhombic one . The transition can

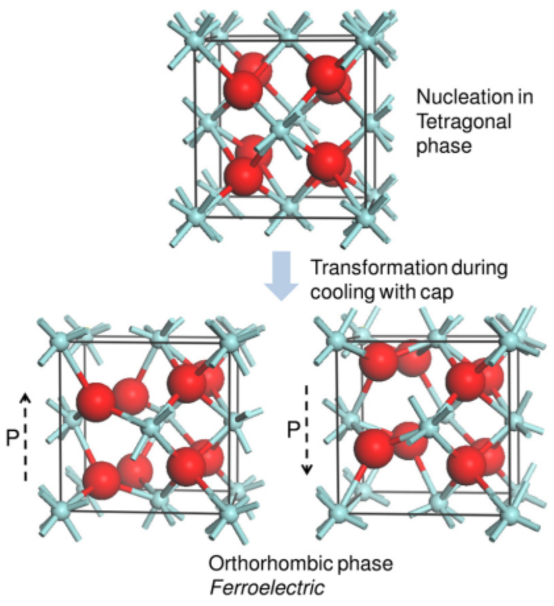
Nucleation in Tetragonal phase
Transformation during cooling with cap
P
P
Orthorhombic phase
Ferroelectric

Figure 1.6. Tetragonal to orthorhombic phase transition. Oxygen atoms are red, hafnium ones are light blue. From Böscke *et al.* 2011 [5].

be appreciated in Fig. 1.7 where two 10 nm thick films of Si:HfO_2 doped with 3% molar content of SiO_2 are compared, one is been grown without capping electrode and the other one with it. Analyzing the samples through x-ray diffraction it is evident as in the former the monoclinic phase is predominant while in the latter the diffraction peaks suggest an orthorhombic crystallization structure.

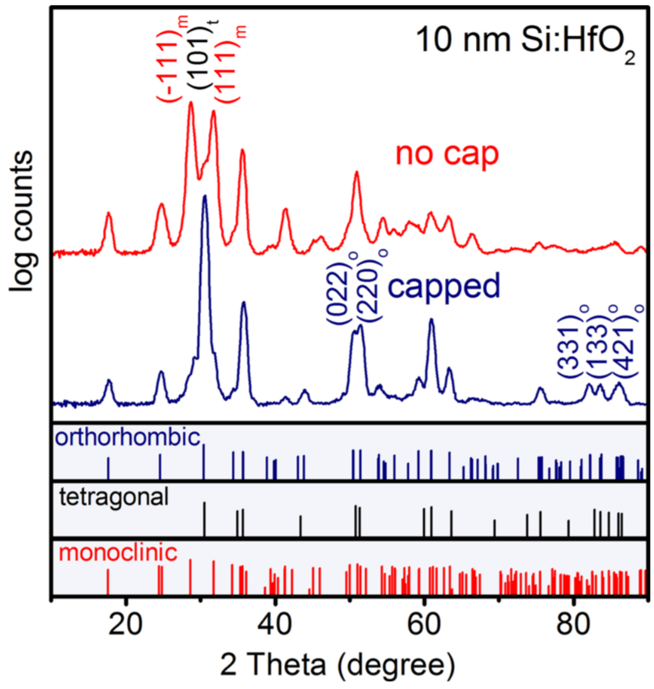

Figure 1.7. Grazing incidence x-ray diffraction measurements of two Si:HfO_2 samples. Crystallization is been induced with and without capping electrode, blue line and red one respectively. SiO_2 molar content is 3%. From Böscke *et al.* 2011 [5].

The cells produced by the martensitic phase transition are still symmetrical but several dopants, such as silicon, yttrium, aluminum, and zirconium [6], are capable of producing a ferroelectric effect breaking the symmetry, resulting in a crystal structure of the space group $Pbc2_1$ [6].
The total amount of doping determines the material response to an external electric field, taking as an example the Zr case we can appreciate in Fig. 1.8 how the simple HfO_2 is purely dielectric, this is due to the predominance of the stable

monoclinic phase. Increasing amount of zirconium the stability of the orthorhombic phase increases showing a polarization response typical of ferroelectric materials, an hysteresis loop. The maximum of the maintained polarization is 17 µC/cm^2 and it's reached when there's an equal amount of HfO_2 and ZrO_2 in the mixture with a coercive field of 1 MV/cm. Increasing further the doping towards pure zirconium oxide the characteristic shrinks at zero bias and exhibiting double-loop typical of the antiferroelectrics materials. In this samples the predominant phase is the tetragonal one [6].

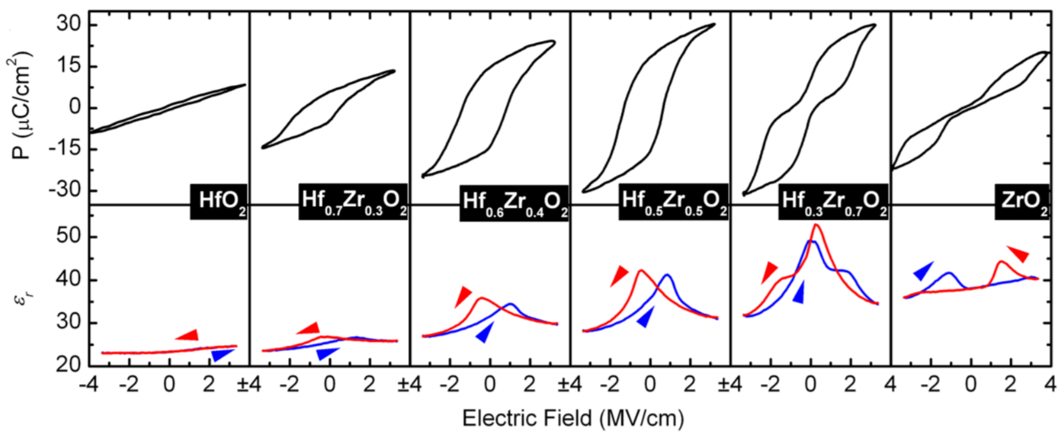

Figure 1.8. P vs. V hysteresis at 1 kHz and small signal CV hysteresis at 10 kHz (50 mV level) of 9 nm thin HfO_2 to ZrO_2 based metal-insulator-metal capacitors at room temperature. From Müller *et al.* 2012 [6].

1.2.1 Technological processes

The technological processes used in thin film deposition are numerous and they can differ a lot in the followed procedure, however they share the same basic specifications: they have to be able to deposit a thin film on a target substrate having control over the thickness produced. A thin film is defined as a layer of material having a thickness in the range from fraction of a nanometer (monolayer thin film) to few micrometers.

Deposition techniques from vapor phase are the most common in hafnium dioxide thin films production. They can be grouped into two macro-categories based om the fundamental method with which the material is deposited:

- Chemical Vapor Deposition (CVD)
- Physical Vapor Deposition (PVD)

The former, CVD, is usually realized by inserting gas precursors into a chamber where the substrate it's present. It is heated to work as a catalyst for a chemical reaction to happen on the surface, this allows the precipitation to grow on the sample. The reaction's byproduct are then extracted by the chamber concluding the deposition.
In the latter, PVD, the material to be deposited is instead broken apart using physical procedure, usually it's heated to the vaporization temperature or bombarded with ions. The atoms ejected in the chamber then lands on the substrate forming a coating.
For the production of HfO_2 thin films the most common used techniques are: Atomic Layer Deposition (ALD) and RF sputtering.

Atomic Layer Deposition Atomic Layer Deposition is a particular type of CVD technique which involves gaseous precursors. In a normal Chemical Vapor Deposition process the reactants are present in the chamber at the same moment while in ALD they are alternate in different moments. The gasses interact with the surface producing a self-limiting reaction, in fact the single phase can proceed until there are reactive sites on the sample's surface.
It is defined a pulse when a precursor is pumped into the reaction chamber, while a purge is when the reactant is removed completely from the chamber's atmosphere before introducing the next one. A complete ADL cycle is composed by the pulse of the first precursor, the purge of it, the second one's pulse and it's relative purge (Fig. 1.9).

Ideally after a single cycle a single monolayer is been deposited on the target surface. ALD performance are described using the growth per cycle parameter instead of the growth rate, typical of CVD which have a more steady growth. The cycle duration depend mostly on two parameters: the precursor pressure and the sticking probability [7], the growth rate can vary drastically on the ALD specific process but is generally slow compared to other thin films deposition techniques. The advantages of Atomic Layer Deposition are mainly the great control and reliability over the thickness of the layer, the purity of the grown crystal and the conformity obtained through this process are remarkable.

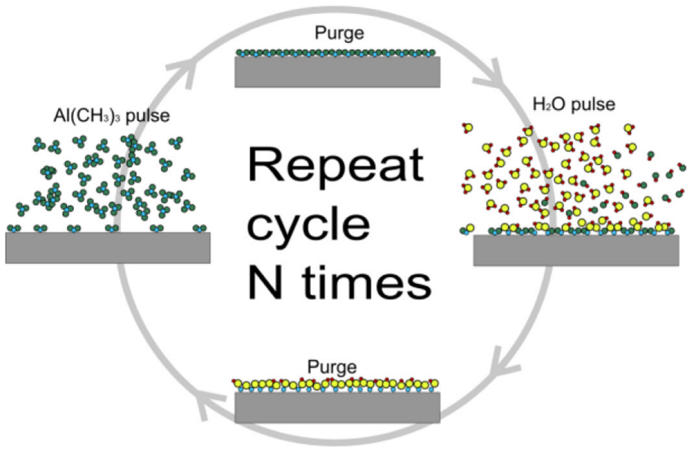

Figure 1.9. Diagram representing an Atomic Layer Deposition cycle. From Säynätjoki, 2012 [8].

RF sputtering The sputtering deposition technique is part of the PVD class. The target is placed in a chamber with controlled atmosphere, on two electrical terminals are placed the sample to be coated, on the anode, and the target to be sputtered, on the cathode. Plasma is then produced in the region near the cathode and a DC voltage is applied between the two terminals to sustain it. The ions produced are then accelerated towards the target. here they create a chain of collisions between the various atoms in the material and if some of them are pushed away from the target with an energy that is higher than the binding energy then they are ejected (Fig 1.10).
The most suitable choice for the gas to be ionized is argon due to a trade off between a mass and cost. A larger mass is desirable to have more kinetic energy transmitted to the material [9].

A disadvantage of this specific deposition technique, DC sputtering, is that if the target material is not a good conductor the ions bombarding the material cannot be discharged. The accumulation of those charges can produce an electric field that grows until the ion are repulsed from the cathode. Because of this reason the DC sputtering is a suitable technique mainly for metal deposition. Radio Frequency (RF) sputtering avoid charge accumulation using an voltage with a fixed frequency of 13.56 MHz [9]. The sputtering, both in DC or AC, can be combined with the use of a magnetron, it creates a magnetic field to keep confined the secondary electrons emitted from the sputtered material. These are kept close to the surface increasing the local plasma density.

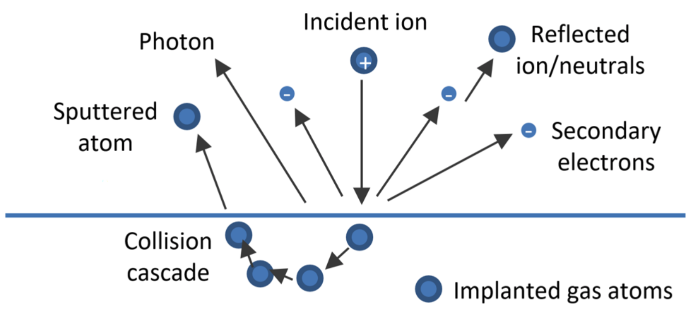

Figure 1.10. Representation of atoms' collision chain during sputtering. From http://www.plasmaquest.co.uk/the-technology/sputtering-basics/.

1.3 Ferroelectric devices

The most important and diffused electronic device is without doubt the Complementary Metal Oxide Semiconductor field Effect Transistor (CMOS FET). Its relevance is due to its great performances united with good scalability and relatively low power consumption if compared with other semiconductor devices.
The increasing push towards portable electronic devices made low static consumption of paramount importance and in this optic the search for a non-volatile memory able to compete with Dynamic Random Access Memory (DRAM) performances it's become an hot topic in research.
Ferroelectric memories are good candidates for this position because of their characteristics: in fact they are, of course, non-volatile, they have low power consumption and write time while having high endurance. They also integrate well in the standard CMOS processes [10]. In particular hafnium dioxide happens to be a particular good choice because it is already used in CMOS as gate oxide, being an high k dielectric.

For more than 40 years CMOS obeyed Moore law, doubling the number of transistors into integrated circuits every year [11], this implied an exponential decrease of the minimum feature size every year bringing the oxide thickness to the order of magnitude of a single nanometer. Using SiO_2 as dielectric the leakage current due to tunneling effect through the oxide energy barrier generated an unacceptable static power consumption.
Since this current is exponentially proportional to the thickness of the oxide and the gate capacitance of a FET device can be expressed as:

$$C = \frac{\epsilon_0 K A}{t} \tag{1.3}$$

where A is the area of the capacitor, ϵ_0 is the vacuum permittivity and K is the relative permittivity of the oxide used.
In order to get the same value of capacitance while reducing the amount of static power consumption became important the concept of equivalent oxide thickness

(EOT) in terms of SiO_2.

$$\mathrm{EOT} = \frac{K_{\mathrm{SiO_2}}}{K_{\mathrm{ox}}} t_{\mathrm{ox}} \tag{1.4}$$

From this is evident that if the chosen material has an higher relative permittivity than silicon dioxide, the same capacitance can be achieved having an higher thickness, therefore reducing the tunnel effect current.
HfO_2 is used in CMOS processes for this purpose, in fact having $K_{HfO_2} = 25$ and $K_{SiO_2} = 3.9$ the gate oxide thickness can be increased by a factor of over six with respect to silicon dioxide [12]. In Fig. 1.11 can be seen the side by side comparison of a SiO_2 and high K oxide gate stack obtained through transmission electron microscopy (TEM).

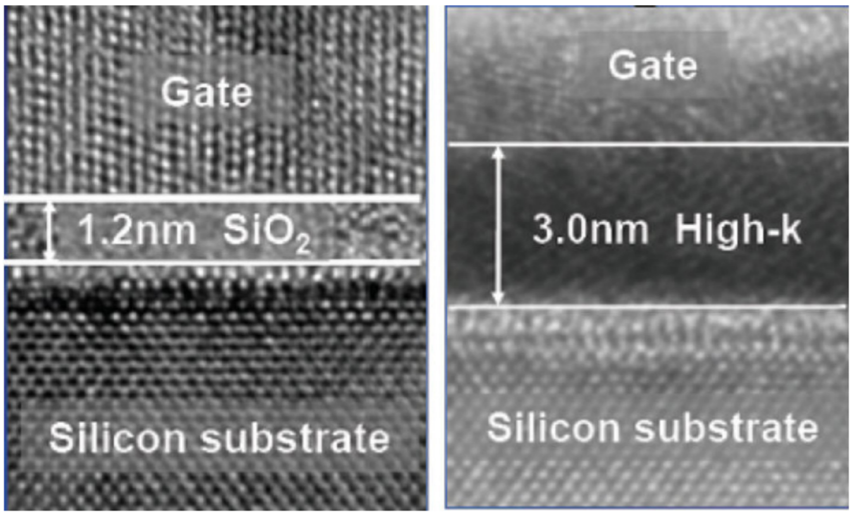

Figure 1.11. SiO_2 (left) and HfO_2 (right) gate stack images obtained through TEM. From Robertson *et al.* 2015 [12].

This key characteristic makes hafnium dioxide a good candidate for integration of ferroelectric devices with the actual CMOS technology, in particular two type of them are important: the ones that uses capacitors such as ferroelectric random access memories (FeRAM) and ones that integrate ferroelectic materials into CMOS technology like ferroelectric field effect transistors (FeFET).

1.3.1 Ferroelectric RAM

Ferroelectric RAM exploits ferroelectric capacitors in order to store information in a non-volatile fashion. The bit is in fact encoded in the orientation of the ferroelectric polarization.

The basic cell of FeRAMs is similar to the one of DRAMs, in fact the most simple cell is constituted by just one transistor and one capacitor (1T1C), its schema is showed in Fig. 1.12. Two operation modes can be distinguished: write and read operations. The former is executed activating the transistor, it charges the capacitor plates producing an electric field on the ferroelectric material. This forces the inner polarization in one of the two stable states that can be encoded as '0' and '1'.
The read operation is performed, again, activating the transistor to force the capacitor's electric field in one specific orientation (e.g. the one corresponding to '0') and reading the current on the output line. If the cell state is already '0' no current is produced, when the state is '1' the rearranging of atoms in the ferroelectric material produces a peak of current that is read by a sense amplifier. The read operation is a destructive process because overwrite the content of the cell, because of that data must be stored back in the cell after reading it.

The speed performances of FeRAM in storing and reading data are comparable to DRAM ones even though the ferroelectric effect is much faster in setting to a stable state. While DRAMs are limited by the time needed to charge the capacitor FeRAMs are instead limited by overall switching delay of the control circuits.
In terms of density they are still comparable due to the great integrability of HfO_2 into CMOS processes, however they are limited by the fact that when the layer become too thin it lose its ferroelectric properties [14]. The major advantage of FeRAMs devices over DRAM is the low static power consumption.
The read/write endurance of 1T1C cell memories is been observed to be in the order of 10^{24} cycles making this devices effectively endurance-free [15]. Most of DRAM power dissipation is due to a process called refresh, the charge accumulated on the capacitor plates leaks through the dielectric material and through the switching transistor. Because of this reason the data retention in the cell is quite short and

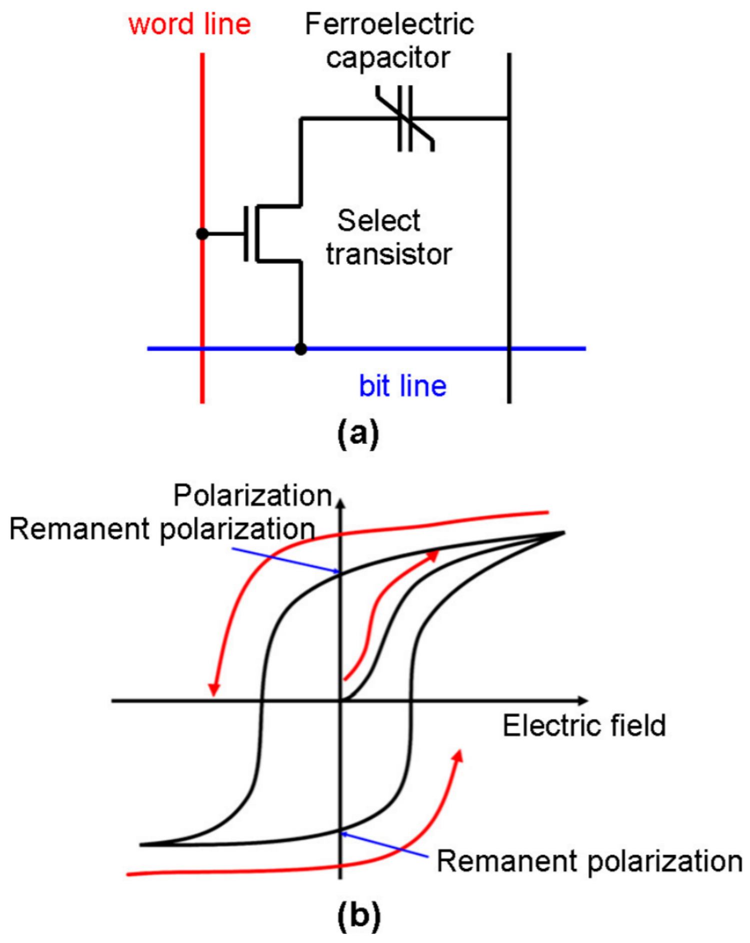

Figure 1.12. One transistor, one capacitor (1T1C) FeRAM cell. (a) diagram of the cell, (b) hysteresis loop and retained polarization. From Ou *et al.* 2020 [13].

the bit must be periodically read and written back to avoid lost of information. In FeRAM, since the information is encoded in the retained polarization, the leakage of charges is not a problem and a refresh operation is not needed, greatly reducing static power consumption.

1.3.2 FeFET

FeFET transistors represent an evolution of CMOS devices with hafnium dioxide as high K dielectric. The oxide layer is doped with zirconium to get the ferroelectric properties and the process differs from the standard one just by two masks [16].
When a voltage higher than the coercive voltage V$_c$ is applied to the gate terminal the intrinsic polarization of the thin layer is changed according to the electric field generated. This produces an accumulation or a depletion of electrons in the channel depending on the state, this results in a shift of the threshold voltage V$_{th}$ between two levels. The low V$_{th}$ level corresponds to a programmed state, or a '1', the high V$_{th}$ level is instead assumed as an erased state, or a '0'. The difference between the two voltage levels is called memory window. The state of the system can be sensed through the transistor drain current amplitude as shown in Fig. 1.13.
The presence of a back bias electrode in this design allows for further flexibility in the choice of the the threshold voltage opening the possibility of programmable logic gates.

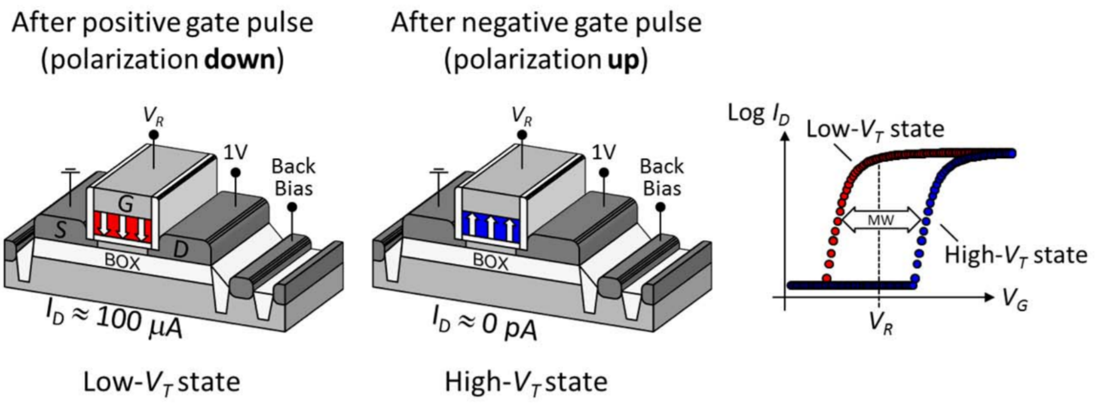

Figure 1.13. A fully depleted silicon on insulator (FDSOI) FeFET representation in both states. On the left its I_D vs. V_R response. From Dünkel *et al.* 2017 [16].

FeFET memories present a non destructive reading, removing the need for a

rewrite operation unlike the FeRAM implementation. They present a smaller elementary cell which can have a competitive area footprint of 0.036 μm^2 but they lack in terms of endurance. FeFET devices have been reported to be able to sustain up to 10^5 write/read cycles (± 3.5 V pulses for 10 µs) [16].
The two main factors of ferroelectric memories' degradation are aging and fatigue of the material. They are both caused by impurities and point defects accumulated in the material but they differ in the condition they develop and in the effects on the material polarization response to an external electric field.
As shown in Fig. 1.14 the former, aging, evolves after time without electrical stress on the material, it presents on the characteristic curve of the material as a pinching of central portion of the hysteresis loop, making the response more similar to the one of an antiferroelectric material. The fatigue effect it's developed after polarization cycles, like memories' write and read operations. It causes a reduction of the total remaining polarization, compressing the loop along the vertical axis [17].

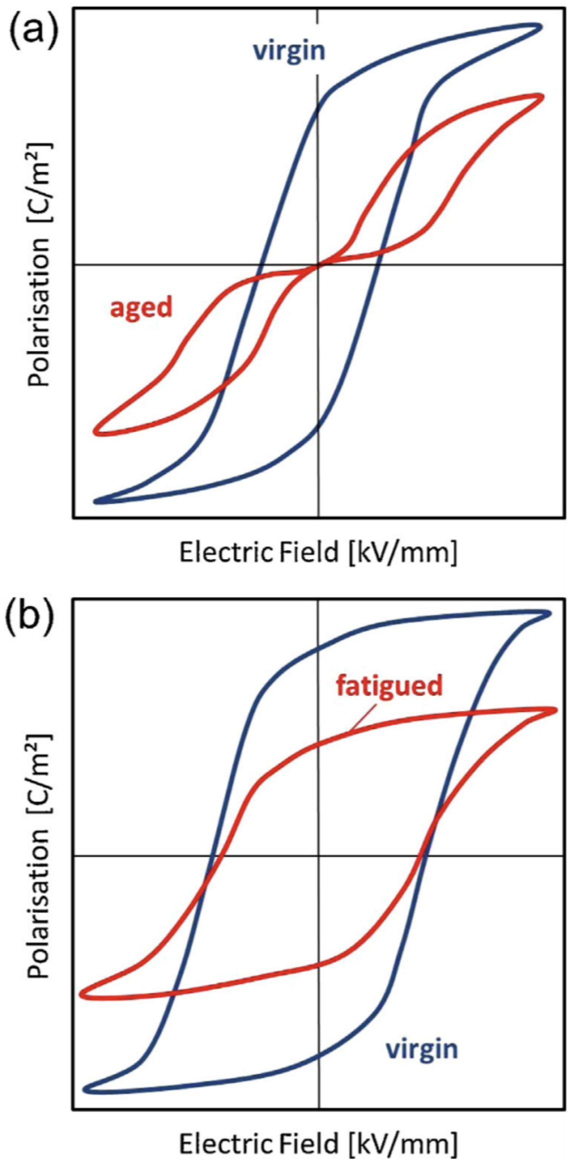

Figure 1.14. Effect of aging (a) and fatigue (b) on ferroelectric P vs. E curve. From Geneko *et al.* 2015 [17].

Chapter 2

Logic In Memory

The increasing popularity of Internet Of Things (IOT) has introduced, in recent years, a shift in computation locality paradigm. From a centralized approach, where a big server is in charge of all the computations, a more decentralized paradigm is rising, pushing towards edge computing. In this case to the object connected to the internet is not only demanded to generate useful data through its sensors but also to perform precomputations and decisions based on local parameters to be later sent to the cloud system.
This has some consequences, including lowering communications' frequency and reducing the amount of data sent through internet, increasing overall speed performances. The drawback is that a great static power efficiency is required, especially because of the poor energy budget that these devices usually have since most of those are portable devices. In fact low leakage current and non-volatile memory are essential to the applications and different new technologies are being investigated in order to solve these issues, among which ferroelectric is a promising candidate. Furthermore, the great cost of communications between processor and memory, the so called Von Neumann bottleneck, has to be minimized.

Because all of these needs a shift in the architectural paradigm is considered: moving the computations inside or closer to the memory can overcome part of the obstacles. In this chapter, after explaining what the Von Neumann bottleneck is, a brief taxonomy of innovative logic-memory paradigms is exposed.

2.1 Von Neumann architecture

Most of modern machines are all based on various evolution of the Von Neumann architecture, inheriting both its best and worst aspects.
These architectures present some key elements: a central processing unit, an internal memory, a mass storage unit and a input and output interface. The central control unit is in charge of two tasks: to perform arithmetic and logic operations together with decoding and executing commands. The memory stores both data and program's instructions. I/O is needed to take input from the external and store outputs on a mass storage unit.

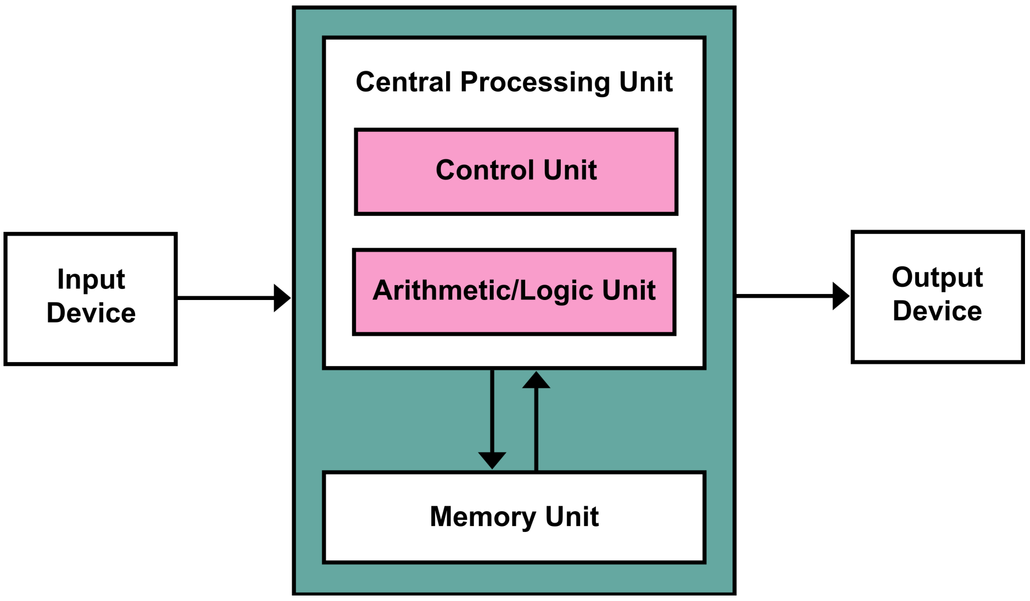

Figure 2.1. Von Neumann architecture. From https://tinyurl.com/y2x4xs5y.

The main characteristic of the Von Neumann architecture is the single bus to communicate with the internal memory, this makes the overall system simpler but has a crucial drawback. In fact the fetch or store of information and data must be sequential, this increments the number of communication needed to perform a single instruction and, of course, the time it takes to execute it increases.
If we take a basic operation like the multiplication as an example, it needs four

elementary assembly operations to be performed:

- load the first variable inside the CPU registers;
- load the second variable inside the CPU registers;
- execute the multiplication;
- store the result back in the main memory;

The CPU has to read each operation from the memory itself, starting from the first load, this is the first communication between memory and processing unit. Then the pointer to the address where the variable is stored inside the memory is read and subsequently the value itself. Only this operation requires three different communications to be performed.
The multiplication instruction needs only to be read from memory since the logic operation is executed inside the CPU.
In the end the store requires again three communications to be performed: one for the instruction itself, one for the address to store the result and one to actually store it.
A simple multiplication is then completed requiring the single data bus to the memory to be used ten times in total, while the address bus is used for every one of the above mentioned read/write operations, requiring ten additional communications (Table 2.1).
The typical communication cost is in the order of 1 pJ/bit/mm while the cost of computation ranges from 1 aJ/bit to 10 aJ/bit, in the case of a 32-bit multiplication the total amount of energy consumed for communication only is equal to $640\,\mathrm{pJ\,mm^{-1}}$, that's equivalent to the 95% of total energy consumption required [18].

The overabundance of processor-memory interactions is the Von Neumann bottleneck and it's a factor that can severely affect performances if not mitigated. It's consequences on those have been also exacerbated by the great imbalance of performances improvement in the last decades between the two types of devices. It's generally both slower and more energy demanding to consult memory than it is to resolve computations.

Table 2.1. Number of memory-processor communication needed for a multiplication operation.

operation	data	address	total
load #1	3	3	6
load #2	3	3	6
multiplication	1	1	2
store	3	3	6
total	10	10	20

2.2 LIM paradigms

The concept of an enhanced memory capable of performing computation or logic operations is called Logic-in-Memory (LiM).
The net separation between processor and memory of the Von Neumann architecture is abandoned to mitigate the effect of its limitations, mainly focusing on trying to reduce the energy requirement.
Various approaches have been investigated and all of them fall under the definition of LiM even if they present very different key features, because of that the need for a classification inside the topic itself is needed.
A rigorous and universal definition of these concepts still does not exists, because of that the taxonomy adopted is briefly exposed.

The classification adopted here is based both on the complexity of the operations carried on by the memory itself and on the integration between logic and memory functions, we can distinguish three cases main cases [19]:

- Processing-in-Memory (PiM);
- In-Memory Computing (IMC);
- Logic-in-Memory (LiM).

Processing In Memory The first straightforward approach is to place a small processor inside or very close to the memory module. Its purpose is to execute small but frequent tasks since it's able to fetch and store data in a fast way because of its proximity, Fig. 2.3 a).
A key characteristic is that the operations carried out by the module are complex, because of that a proper processor is required, complete of an instruction set, a control unit and an instruction register.
The main CPU can then offload part of the task to the local processor reducing communication frequency but increasing the memory footprint.
Memory and processing devices are close but still both logically and physically separated.

An example of the above class is showed in the work of Pugsley *et al.* 2014 [20] where PiM applications for big data distributed computation are investigated.
The approach used is the one of Near Data Computing, where the processor is placed very close to the memory elements in order to reduce both latency and communication energy consumption. The application involves a great level of parallelism to handle the huge workload of the processes, to further increase performances on the local level technique of 3D stacking are exploited.
The Hybrid Memory Cube from Micron is been used since it stacks DRAM dies on logic circuits connecting the two of them with Through Silicon Via. These are vertical connections created through the various insulator layers of the die formed by a hole filled with conductor material. Exploiting the vertical dimension they ensure a short and narrow connection, permitting a superior bandwidth with respect to DDR3 and DDR4 RAMs [20].

Given the complexity and diversity of the operations required by the application requires the use of energy efficient general purpose processors.
This PiM architecture is been observed to reduce execution time by a factor of 15 and system energy by a factor of 18 [20].

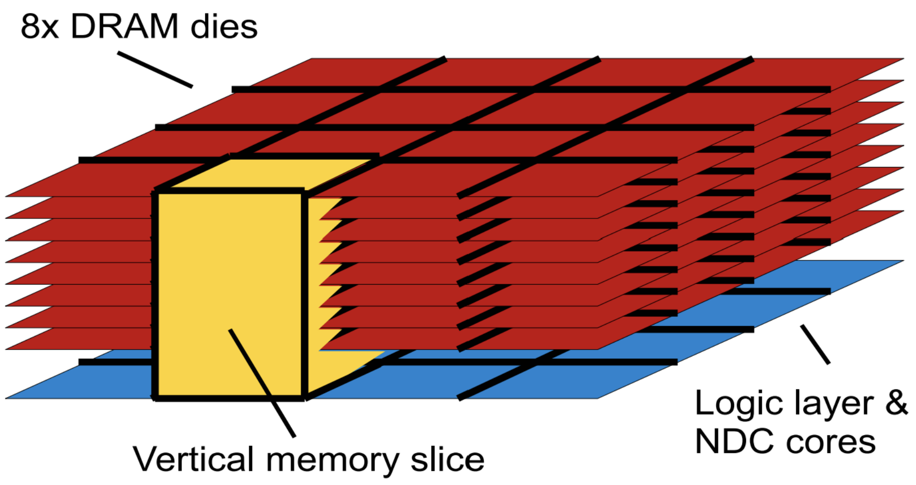

Figure 2.2. Representation of a DRAM banks stack connected to the logic layer. From Pugsley *et al.* 2014 [20].

In Memory Computing The memory module has integrated some elementary computing operations, in this case the CPU asks for an operation between two or more stored variables and the memory replies with the result, Fig. 2.3 b).
The tasks are smaller and the amount of communications between memory and processor to send control signals are higher with respect to the previous case but memory area is reduced.

Logic In Memory The properly called Logic In Memory differs from the two previous definitions because it performs operations without the need to add dedicated logic operators but only with existing memory resources.
This can be subsequently subdivided into two categories: coarse-grain and fine-grain LiM. The former, *coarse-grain* Logic In Memory, has the peculiarity of not executing logic into the memory at all. Frequent tasks' results are stored into the non-volatile memory to be read; this is placed very close to the processor in order to obtain fast access. For complex functions are available different strategies to remove some stored elements, while this greatly reduce the area footprint further elaborations on the results may be required depending on the case this removes the need for intermediate communications.

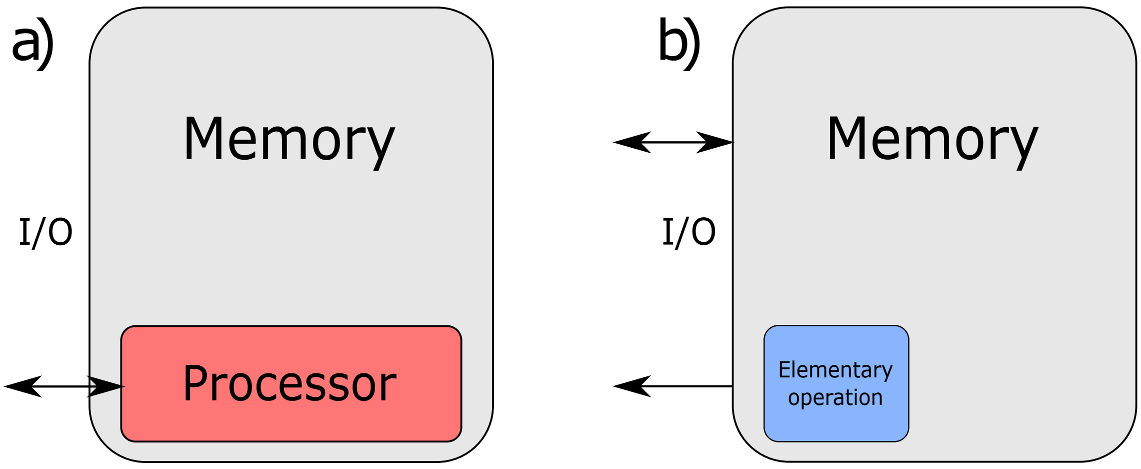

Figure 2.3. a) Processing In Memory representation, the processor is placed inside the memory device but they are different entities. b) In Memory Computing representation, logic and data storage are still separated but the former is simpler and does not require a control unit.

This non-volatile lookup table (LUT) approach removes the needs for operands to be sent to the processing unit and having the same speed of a L1 cache it mitigates the effect of the Von Neumann bottleneck.

The latter category is *fine-grain* Logic In Memory. Here very small blocks of memory are connected to logic circuits to achieve new functions.
Those components can well suit applications in which there are a set of fixed parameters that are locally used to perform frequent operations like in Finite Impulse Response digital filters or convolutional neural networks.
An example of integration can be few non-volatile flip-flops added to input ports of logic gates to perform re-programmable logic or, more specifically to the FeFET application in fine-grain Logic In Memory, the exploitation of the intrinsic memory capability of the device to perform logic operation between two input bits, one stored in the ferroelectric capacitor and one sent to the transistor's gate.

An example of coarse-grain LiM is the work presented by Chen *et al.* in 2018 [21].

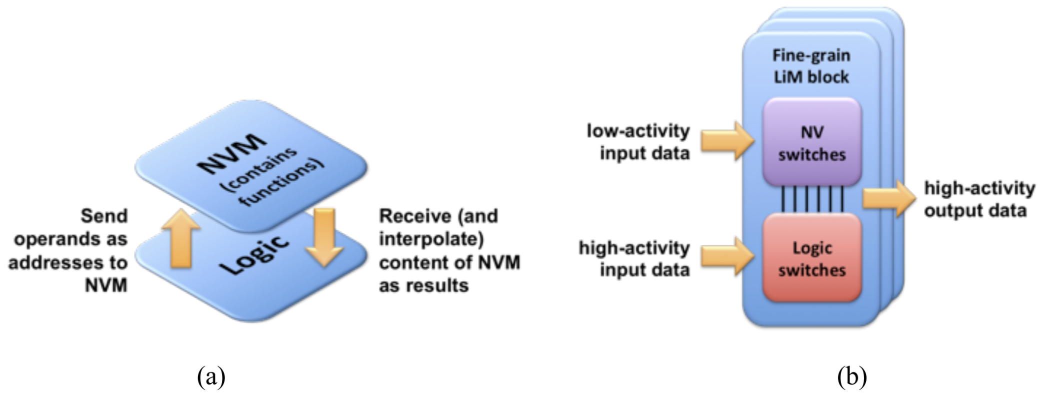

Figure 2.4. Non-volatile Logic In Memory representation. (a) coarse-grain (b) fine-grain. From O'Connor *et al.* 2018 [19].

The proposed architecture exploits an array of non-volatile ferroelectric memory explicitly designed as a look-up table.
The elementary cell proposed is composed by a single FeFET (Fig. 2.5) and a custom write/read scheme is been designed together with the architecture.
The proposed LUT leverages the great integration of FeFET with CMOS processes to obtain an highly area-efficient circuit with respect to other non-volatile technologies. It also takes advantages of the near-zero current of FeFET in the off state, one of the key characteristic of the technology.
It's been proved that for 6-bit inputs LUT the area-power-delay product is 6.2× better than designs based on static Random Access Memory (SRAM), 3.9× better than Magnetic Tunnel Junctions (MTJ) LUTs and 2× better Resistive Random Access Memory (RRAM) LUTs [21]

An example of fine-grain LiM are the FeFET reprogrammable logic gates presented in the work of O'Connor *et al.* [19] depicted in Fig. 2.6.
The basic principle exploited is that one input bit is stored into the ferroelectric layer as the polarization direction, it's written sending a write/erase pulse to the FeFET gate. The other input bit is sent to the gate as a readout voltage low enough to not change the former.

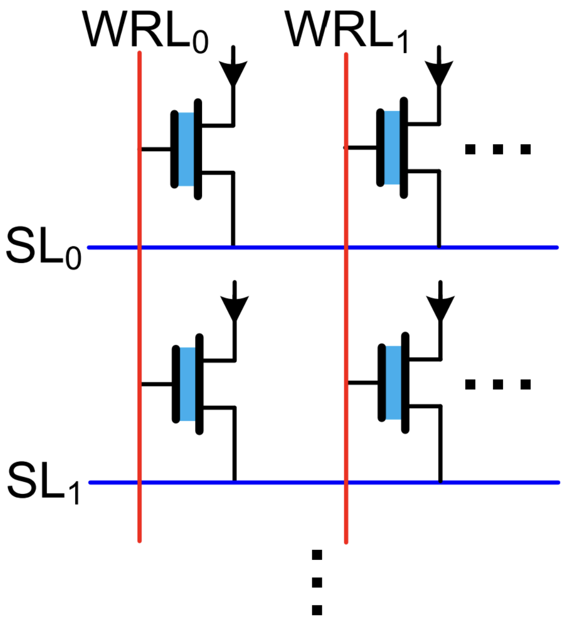

Figure 2.5. Array of elementary LUT cells. From Chen *et al.* 2018 [21].

The state in which the device is set shifts the threshold voltage, changing the response to the input. In the '0' state, or high resistance state the threshold voltage is higher and represented as a solid line in Fig. 2.6 b), in the '1' state, or low resistance state, it's lower and represented with a dashed line.
A further degree of freedom is introduced by the back bias voltage that can shift all the response, changing the logic behaviour of the gate, keeping the same readout voltage levels.
In particular the scheme exposed in Fig. 2.6 a) represents a NAND/NOR gate: in one state of the back bias we get the $I_d - V_g$ curve shifted toward the positive voltage axis and the FeFET produces an AND response. In this case the output is coded into the amplitude of the I_d current, if the transistor is on and the I_d current is not negligible and it is considered an on state or a '1', in the opposite case is an off state or a '0'.
The presence of a pull up resistance act as an inverter while transducing the output

again into a voltage: if the transistor is off the output terminal is no longer connected to the ground and the resulting output voltage is dictated by the pull up circuit to V_{DD} setting a '1', when the transistor connects the output terminal to the ground the output voltage drops to almost zero resulting into a '0' state.
These results are showed in table 2.2.

Table 2.2. Truth table for NAND/NOR 1-FeFET logic gate. When back bias is set to '0' the logic function implemented is NOR, when '1' is NAND.

	back bias	A	B	I_d	V_o
NOR	0	0	0	0	1
	0	0	1	1	0
	0	1	0	1	0
	0	1	1	1	0
NAND	1	0	0	0	1
	1	0	1	0	1
	1	1	0	0	1
	1	1	1	1	0

It is evident how the proposed taxonomy divides the subject into categories based on the complexity of the operations handled by the memory, ranging from complex macro tasks proper of PiM to the bit by bit operations of fine-grain LiM, but at the same time the level of integration between logic and memory components changes as well, ranging from completely different devices placed close to each other to a very intimate mix of the two.
The enormous variety in those paradigms reflects itself into infinite degrees of freedom in the design of Logic In Memory devices and an efficient way to compare those different approaches in a meaningful way is of paramount importance when choosing the best architectural structure for a specific target operation. The proposed benchmarking platform tries to fill this gap, offering a fair way to analyze different approaches to the same problem in a single tool.

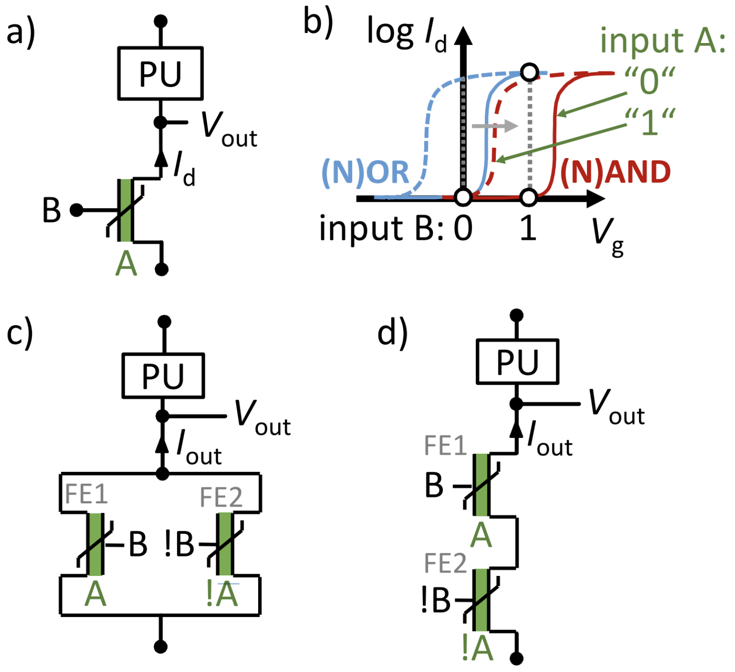

Figure 2.6. a) 1-FeFET reprogrammable NAND/NOR logic gate schematic with its basic functionality depicted in b). c) 2-FeFET XOR logic gate. d) 2-FeFET XNOR logic gate. From O'Connor *et al.* 2018 [19].

Chapter 3

Benchmarking platform design

The great number of possibilities in LiM architectural development, combined with the different technologies emerging in the last decade, gave rise to an overwhelmingly vast design space in which is easy to get lost or to miss the optimal solution for a specific use case.
Because of this the need for a trustworthy and reliable method to compare different approaches, even before the experimental step, is of paramount importance in the design of a LiM architecture.
This project proposes a platform that offers itself to tackle this impressive task.

The result here detailed is incomplete and groundbreaking numerical results are still absent. Anyway, the ones that are present prove its capability of becoming an important addition in the tool belt of researchers focusing in the field.
The platform design is based upon three fundamental concepts:

- abstraction;
- scalability;
- modularity.

The first, abstraction, is needed for being able to simulate systems that are fundamentally different between them. High level system simulations on the components are performed in order to extract performance estimations to be evaluated. These are based on physical parameters typical of the technology node taken in exam.

The second, scalability, is important for the platform to be able to accommodate different benchmarking specifications based on the chosen use case.
The latter, modularity, permits to create custom simulations with only the components chosen for the analysis and to include in the future new features and functionalities.
The platform is developed with extreme flexibility in mind, for future compatibility and to be able to satisfy the maximum number of scenarios required by the eclectic nature of LiM architectures.

The platform is divided in three parts: first the benchmarking algorithm is chosen and, if needed, translated in a format that the platform can elaborate. The input syntax is defined and later discussed in more details.
The instructions are then read by the actual platform which performs the simulation and returns the figures of merit. In the end a final stage is performed in which the results are cleaned of unnecessary information, confronted if needed with previous simulations and plotted.
The interfaces formats between those stages are rigorously defined, in this way the single parts can grow independently one from the others, granting the modularity of the complete system.

The platform, given its high level of abstraction, is able to switch between different technologies or different implementation of the same module. The parameters received through various operational cards capture the physical aspect of the simulation, allowing to include all of them in the same framework in a reliable manner.

3.1 Platform anatomy

In this section a more detailed overview of the platform structure is given, focusing on the benchmarking algorithm and the results elaboration. The simulation platform will be thoroughly discussed in section 3.2.

3.1.1 Overview

The whole system is divided into three parts that share inputs and outputs through written files. The three main macro-component can be seen in Fig. 3.1 and they are: benchmark, simulation platform and output handling.

The first, benchmark, consists in a text file listing all the operations that the platform needs to perform, resulting in a very basic *compiled* version of the chosen benchmark in a syntax comprehensible by the simulator.
The following phase is the actual simulation, the executable parse the instructions and computes the results while recording power consumption, latency and others figures of merit. The platform is implemented utilizing the SystemC library for the language C++, this is a full-fledged Hardware Description Language (HDL).
The choice of an HDL is mainly due to two reasons: it permits a more flexible degree of simulation accuracy while guarantying the modular nature of the platform. In fact a single module can be considered at a very high system level, just emulating the logical behaviour and estimating general parameters, or the description can descend into further details, using a Register Transfer Level (RTL) representation. This gives the opportunity to test and confront different implementations of the same logical unit.
The output of this stage is a file reporting the traces of important signals present in the given simulation.
The last stage takes as input those traces and, after a decoding step, it plots them. It presents a feature which permits to display more simulation results in the same plots, facilitating the confront between architectures.
The graphs produced are stored in a specific repository.

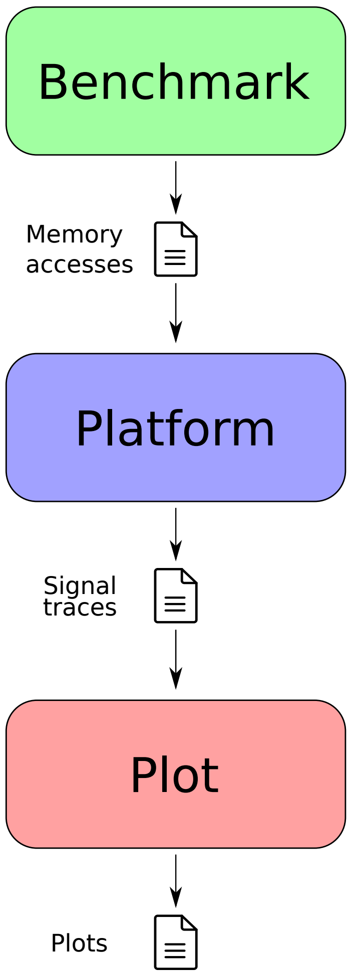

Figure 3.1. Schematic representing the different steps involved in a single simulator run. Memory accesses are in txt format, traces in vcd format and plots in svg format.

3.1.2 Benchmark

Arguably one of the most important aspects of a simulation platform are the benchmark that it's capable of running.
Since the platform is designed as a coprocessor it must receive memory accesses and operation instructions from the main computing unit, those information are stored into a file that is later given as input to the simulation program.
The platform is totally independent from the benchmark origin system, this permits a further degree of flexibility because, as long as the operation are coded in the

syntax understood by the platform, the benchmark is guaranteed to be executed as intended.
Traces can be generated by custom made programs, by industry standard programs at runtime or they can be extracted through compiler instrumentation.
The current encoding is been kept as linear and simple as possible to have the best performances and to maintain forward compatibility with further enhancements of the platform.

At the current state a Python script is in charge of creating the instructions for the LiM accelerator, the algorithm chosen is squared matrix multiplication. This was the obvious starting choice because of his paramount importance and elevate frequency.
The basic functionality of the script is to take as input a file called `matrices.dat` containing two signed integers matrices and to write the operations needed for executing the multiplications in the file `matrix_multiplication_inputs_[nwb/wb].txt`.
The script can takes as input different optional parameters:

- `--nwb`: this parameter set the output as a No-Write-Back (NWB) process (default mode);
- `--wb`: this parameter set the output as a Write-Back (WB) process;
- `--new`: this parameter ask to use new matrices that are stored in the `matrices.dat` file;
- `--d`: this parameter set the dimension of the square matrices through an integer.

The script is designed to make the LiM hardware perform the whole computation and store the results. This is not the most efficient way in terms of energy and speed but it was necessary for two main reasons: first of all the optimal implementation would consider the synergy with a main processor that is absent in this stage of the project, moreover this configuration makes possible the computation of a numerical result, giving us the opportunity to extrapolate figures of merit for the error introduced by the device.

Because of this design choice the resulting instructions file can be divided into three main parts: the first is the storing of the input data inside the memory, the second is the operations needed to perform the calculations and the third is the reading of the obtained results. The last one is needed for error calculations and for debugging purposes.

The output contains a single instruction per line, implemented in a simple syntax. The operation needed is coded into one character and it's followed by other information needed by it. A comprehensive list of available operations is present in table 3.2, .
There are three main templates for benchmark instructions: one for NWB operations, one for WB ones and the last is reserved to memory I/O.
NWB instructions have a total of two parameters representing the operation operands, they are separated from themselves and from the operation character with a space. Taking as an example a NWB addition between the numbers 5 and 7 it would be encoded as:

$$\underbrace{a}_{\text{operation}}\ \underbrace{5}_{\text{in 1}}\ \underbrace{7}_{\text{in 2}}$$

If the instruction is WB an additional input parameter is provided representing the platform memory address in which the result must be stored at the end of the process.
A WB multiplication between 20 an 14 that must be stored in the 1340-th memory position would be encoded as:

$$\underbrace{M}_{\text{operation}}\ \underbrace{20}_{\text{in 1}}\ \underbrace{14}_{\text{in 2}}\ \underbrace{1340}_{\text{address}}$$

Read and write memory operations are simpler because they need less parameters: the former instruction needs just the memory address to be read while the latter requires a target address together with data to be stored.
The memory write operation is unique because has two different operating modes that can be detected at simulation runtime. The first one is the default writing operation, the second is to store back the last output of the simulation platform.

This is a feature implemented to permit the platform to reuse intermediate results and to carry into further computations errors introduced by the system. It's strictly related to the absence of a main CPU in the simulation.
A recap of the different templates is present in table 3.1.

Table 3.1. Table summarizing the different benchmark instruction templates. *Op* stands for operation character, *in1* and *in2* for the input operands while *add* for address.

	Templates
NWB	*op in1 in2*
WB	*op in1 in2 add*
read	*op add*
write	*op* [*add*]

Table 3.2. Table of character used to encode the different possible operations inside the platform.

Coding character	Operation description
w	Write operation
r	Read operation
a	Addition NWB
A	Addition WB
m	Multiplication NWB
M	Multiplication WB
i	Interpolation NWB
I	Interpolation WB

3.1.3 Output analysis

The output analysis is a mandatory step to convert the simulation results into a more significative and readable format. The platform's output consists of a Value Change Dump (vcd) file format which is a non coded text file in which the state changes of different signals are stored. The platform signals considered as outputs, together with the debug signals if needed, are set to be logged on that file.
The plotting scripts is divided in three main parts: first of all the vcd file is parsed, then total energy and total power consumption are computed, in the end data are plotted and saved.
The script accepts three types of parameters:

- `-f`: this parameter set the path of the vcd file to process and the prefix to assign to its graphs;
- `--cmppower`: this parameter indicates to perform, if multiple files are present, the plot with the various total power consumptions;
- `--cmpenergy`: this parameter indicates to perform, if multiple files are present, the plot with the various total energy consumptions.

The data interpreting step consists of parsing the vcd file mapping the encoded key to the signals name. After having acquired the initial values for those it saves couples of data containing its value for any given unit of time. Value zero for the time axis corresponds to the beginning of the simulation. These couples are stored in various array, which are needed for the plots.
All signals present are taken into account for the plots but only if they belong to two categories: boolean types and real number types. Buss signals are ignored due to the difficulties to represent and compare different values with each others.

The simulation platform is designed to always output two main signals: dynamic energy consumption and static power consumption.
These two are used to compute the total energy and the total power of the simulation.
At first total static energy consumption is computed, it's expression at time i is:

$$E_{\text{st,i}} = E_{\text{st,i-1}} + \Delta t \cdot P_{\text{st,i-1}}$$

where $E_{\text{st,0}} = 0$, Δt is the minimum time step possible in the simulation, it's value is set to 1 ps. $P_{\text{st,i-1}}$ is the static power consumption at the time step $i - 1$.
Then total energy consumption is computed as:

$$E_{\text{tot,i}} = E_{\text{st,i-1}} + E_{\text{dyn,i-1}}$$

where $E_{\text{dyn,i-1}}$ is the dynamic energy consumption at the time step $i - 1$. Lastly the total power consumption is computed as the incremental ratio of E_{tot}:

$$P_{\text{tot, j}} = \frac{E_{\text{tot, j}} - E_{\text{tot, j-1}}}{\Delta t_j}$$

where $\Delta t_j = 5 \cdot \Delta t$. This is due to the fact that the delay times of the components analyzed so far are at least of the order of 1 ns, the use of a bigger time step introduced a negligible accuracy error while greatly increasing the program performances.

The last step is to perform the plots of all the signals found in the vcd file. All the graphs are plotted against time and saved in Scalable Vector Graphics (svg) file format inside the *graphs* directory.

3.2 Simulation structure

The heart of the benchmark platform is composed by an executable designed to perform high level simulations of a generic LiM hardware accelerator. It's structure permits it to remain as flexible and modular as possible while still being able to faithfully report results for different architectures and technologies.
The project is been developed into C++, using the SystemC library for two main reasons: first of all is repurposing of already existing code and, most importantly, to permit the integration of RTL level description of modules. This is especially important to be able to implement a full fledged main computing architecture, permitting in this way to extensively evaluate the full spectrum of LiM hardware accelerators interactions.

The aim of this tool is to offer a fair method to compare intrinsically different technologies and architectures in order to fasten the exploration of the design space for researchers in this field.
This is accomplished by separating the architectural implementation from the hardware used. At runtime an operation card is loaded and used to perform the computations, this is a file containing the necessary parameters for each given technology and they are stored in a library fashion.
Swapping between different cards permits to compare the performances of the same architecture against all the available technologies, even different technology nodes can be implemented to have a finer level of detail.
Those parameters can be extracted from literature, low level simulations or from measurements and they can be updated without having to modify the program inner parts.
The architectural level is organized to be as modular as possible: all the components designed are available in each simulation. Different configurations are defined at runtime to be able to perform different tests in a batch of runs. At the moment only two architectural paradigms are implemented (WB and NWB logic in memory) but the platform is designed for forward compatibility and extensions.

The program is also intended as a test bench for new technologies: comparing them side by side with more mature and developed ones their strength, but most importantly their weaknesses, can be unveiled more easily. The prospect is to report back to hardware research closing a feedback loop that would speed the practical implementation up, making at the same time better devices and choosing the most effective application.

The system at the current state focuses on delivering figures of merit for energy and power consumption together with overall latency and error with respect to the expected result. This is just the basic set of output and will be expanded in the future, since every internal signal can be set as an output it can be easily accomplished.

3.2.1 Architectural design

In this section the overall structure of the simulation platform is explained referring to the target architectures. The detail of the single modules will be analyzed in details below.

In this first stage of the project the high level simulation focused mainly on two types of Logic-in-Memory architectures:

- No-Write-Back;
- Write-Back.

They both belong to the LiM family of architecture, explained in chapter 2.2, but they differ in one key aspect: the former returns the computed result to the main computing architecture for further manipulations. The latter stores it back without having to receive additional instructions from the external, avoiding additional cpu-memory communications, a schematic representation of both is shown in Fig. 3.2.
The simulation platform, after some design iterations, it's been settled to a generic configuration able to perform both architectural paradigms even in a mix configuration, performing both types of operation in the same run. The internal macro-structure is organized around the module representing the memory and its showed

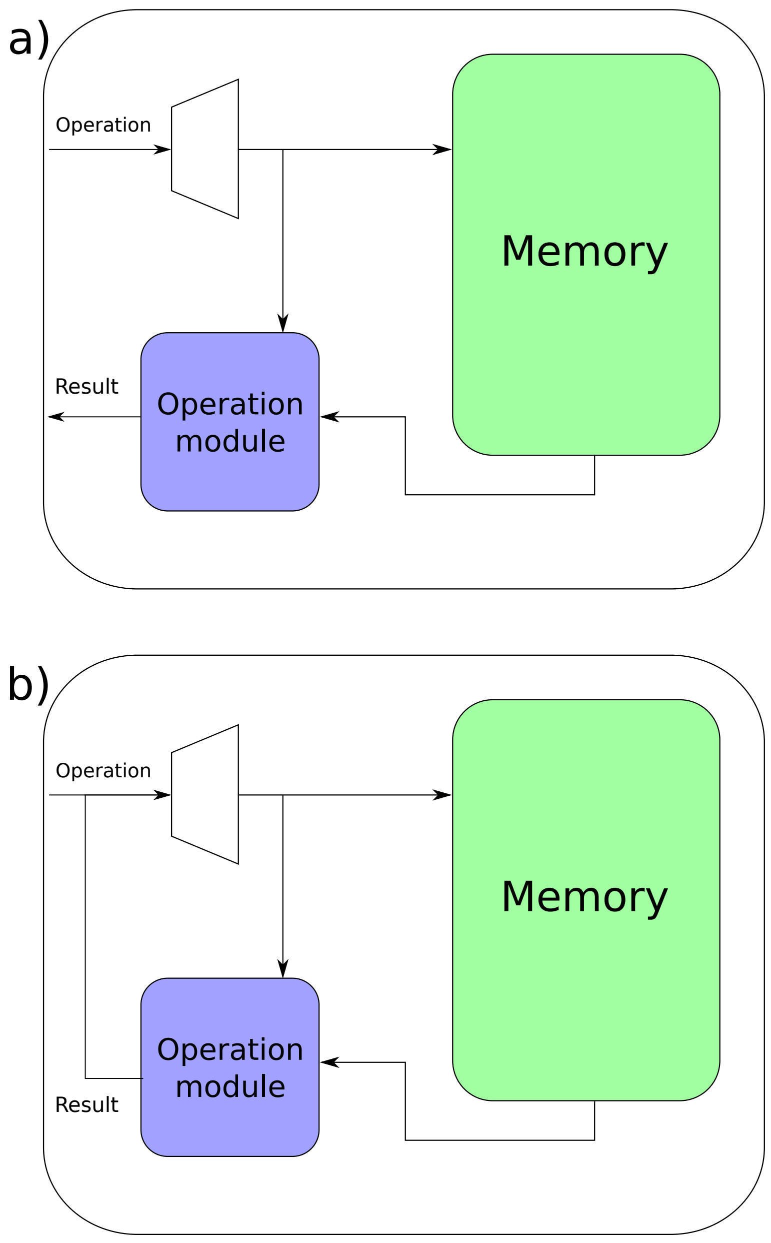

Figure 3.2. Diagram representing: a) No-Write-Back (NWB) and b) Write-Back (WB) architectures.

in Fig. 3.3.

At the moment a pure coarse grain approach is implemented, because of that no logic is performed inside the memory itself. The operations are carried on in separated components instead, leaving the memory cell structure unaltered. It's important to underline that this division is purely logical, as long as the components are not intertwined the physical location of the components in the final dies impacts only on the communication energy consumption. This aspect is neglected at this stage being that communication in the same chip are much smaller than communication between the CPU and memory or than the energy needed to perform operations.
The overall structure implements asynchronous communications. This allows for an easier individuation of bottlenecks in terms of latency.

Incoming instructions are interpreted by a decoder, it's function is strictly related to the high level nature of the simulation. Using an instruction decoder would produce a minor impact when new modules will be added to the platform, resulting in greater flexibility for further extensions. At the same time the architecture is not tied to a single configuration as would it be if the input logic would have been described at RTL level. The input logic power consumption and overall latency are negligible with respect to the memory ones and very dependent on the architecture so they are not been taken into account at the moment.

Instructions, converted into control signals, reach the memory which performs only read and write operations.The data extracted are then redirected to the output or to operational modules based on the received inputs.
Depending on the instruction performed a multiplexer (MUX) choose where to route the operands. The operation now available to the accelerator are:

- addition;
- multiplication;
- multiplication via interpolation.

Each one of them can be performed in both WB and NWB mode with the benchmark instructions showed in table 3.1.

After the computations are completed the output is sent to the output manager component. It is a purely logical component as the decoder, it doesn't concur to the performance estimation but it permits to implement the write back logic. Similarly to the input decoder it's contribution is neglected for the sake of abstraction and flexibility.
This module is capable, in the case that the operation requires a write operation back to memory, to raise the memory control signals. The address sent together with the instruction is used to store the final result correctly.

The simulation executable can take two main input parameters:

- `-b` is followed by the path to the benchmark file to parse for instruction;
- `-n` is followed by the prefix for the output file referring to this simulation.

In addition to those explicit inputs the platform exploits other parameters that can be overwritten using environmental variables before runtime. Those are related to the modules and will be explained in details in section 3.2.2.
The operational cards used by modules to define low level parameters are stored in specific files and they are loaded at run time.

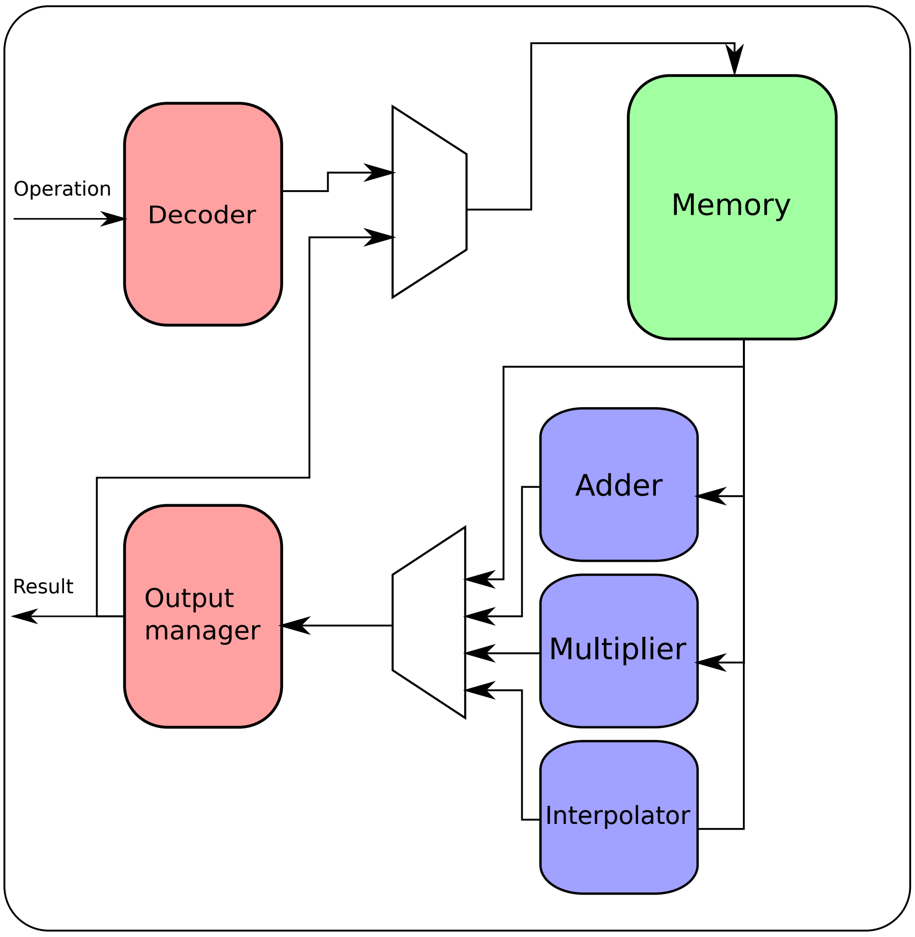

Figure 3.3. Diagram representing the internal structure of the platform. In green is marked the memory module and in blue are the operational modules, components concurring to energy estimations. Non-operational modules are in red, their purpose is to ensure a correct simulation and have no performances associated.

3.2.2 Modules

Inside the simulation platform are present various modules with different characteristics They can be grouped into three main categories: operational modules, non-operational modules and the memory. Operational modules are high level representations of hardware components that perform logic on input data and return a result, together with performances estimations. The second ones, non-operational modules, are components that take the place of input nd output interface logic. Figures of merit deriving from those are considered negligible with respect to those resulting from memory interrogation and CPU-memory communications. Furthermore the design of those components is heavily architectural dependent, resulting in an unnecessary step in the early phases of the project.

All modules present input and output signals, control signals and configuration parameters that are deeply connected to the nature of the module. All of them will be discussed in details in the following sections.

Non-operational modules

Non-operational modules category comprehends:

- input decoder;
- output manager.

The former is essential for the correct interpretation of the input signals. At its core there is a thread that reads the benchmark file. Every line is parsed and for each command received the correct set of control signals is dispatched. This thread is sensible to all the `ready` signals of other modules. When those are set to an high value it means that the component is no more busy and it can handle the next operation. This prevent synchronization issues given the asynchronous nature of the platform.
The energy consumed by address communications to the platform is also estimated into this component and reported into the output signal `address_energy`. The default parameter used for address energy estimation is 1 pJ/bit and it's value is

bounded to the environmental variable ENERGY_ADDR [18]. It is important to underline that this energy consumption is not produced by the logic simulated by this module, it is calculated here for a matter of convenience.

Table 3.3. Table summarizing configuration, input and output parameters of the input decoder module.

env. variables	`ENERGY_ADDR`	Address bus energy consumption per bit
	`WORD_SIZE`	Bit number of memory word
input signals	`ready_mem`	Memory ready signal
	`add_ready`	Adder ready signal
	`mul_ready`	Multiplier ready signal
	`interp_ready`	Interpolator ready signal
output signals	`on_off`	Signal to turn on or off the memory
	`read`	Memory control signal to start read operation
	`write`	Memory control signal to start write operation
	`mul_operation`	Multiplier signal to start operation
	`add_operation`	Adder signal to start operation
	`interp_operation`	Interpolator signal to start operation
	`out_operation`	Output manager signal to start operation
	`mux_select`	Selection signal for memory input MUX
	`mux_select_address`	Selection signal for address MUX
	`addr`	Address bus
	`data`	Data bus
	`out_select`	Signal to discern NWB from WB operations
	`address_energy`	Address energy signal

The output manager is in charge of routing the output data depending on the type of operation, which is communicated through the `select` input signal. In case of a NWB instruction it is set to ’0’, the output is redirected to the external through the bus named `external_output`. In case of value ’1’ it is rerouted to the memory using `memory_output`. This module can also drive the control signals to write into memory.
Another important function of this component is to apply the rounding logic. The connections inside the platform are fixed to a 32 bit width and because of that when

the output is stored or returned it must be adjusted to simulate finite precision arithmetic. In case of overflow the result is capped to the maximum or minimum number representable by the number of bits defined by `WORD_SIZE`, implementing a saturated arithmetic. In every other occasion the number is truncated.

Table 3.4. Table summarizing configuration, input and output parameters of the output manager module.

env. variables	`WORD_SIZE`	Bit number of memory word
input signals	`input`	Input data to the module
	`select`	Signal to discern NWB from WB operations
	`output_operation`	Signal to start the operation of the module
output signals	`memory_output`	Output data towards the memory
	`external_output`	Output data towards the exterior
	`memory_write`	Memory control signal to start read operation
	`mux_select`	Selection signal for memory input MUX

Operational modules

The operational modules are the one performing computations on the data stored in the memory. They are three at the current state:

- adder;
- multiplier;
- interpolator.

Combining the three of them with the memory it is also possible to perform Multiply And Accumulate (MAC), even if it's not implemented as an explicit input command. The various elementary instructions have to be sent in order to perform it.
The first two, adder and multiplier are very similar having as only difference the arithmetic operation performed. They present one input bus from which operands are acquired, this is due to the memory structure having only one output bus forcing a two read cycle to obtain the desired data. As soon as the second number is

acquired the operation is performed asynchronously and, after the logic latency is elapsed, the result is exposed on the output bus. The operation is simulated on a high level and no RTL simulation is performed. Energy and time parameters are set by an operational card and can be changed independently from all the others to perform parametric simulations if required.

A thread is instantiated with the component, this is sensitive to two signals: `add_operation` and `load`. The former communicates to start the operation with it's positive front, the latter that the input data is ready to be sampled in order to get the correct information.
Inside the same thread the energy parameters are updated and then exposed through a signal representing a real number.

Table 3.5. Table summarizing configuration, input and output parameters of the adder and multiplier modules.

env. variables	`WORD_SIZE`	Bit number of memory word
input signals	`add/mul_input`	Input bus
	`add/mul_operation`	Signal to start the operation of the module
	`load`	Signal to load the input data
output signals	`add/mul_result`	Output data bus
	`add/mul_energy`	Total energy consumption of the module
	`add/mul_ready`	Signal for module availability
operational card	`n_bits`	Module precision
	`energy_bit`	Energy consumption per bit
	`latency`	Time delay introduced

The last operational module is the interpolator. It performs multiplication between two signed integers operands by reading a partial result from a sparse LUT placed in memory and recovering the result through a bilinear interpolation.
The number of bits of the operand is defined by `INPUT_RANGE`, while the degree of sparsity of the LUT is defined by the `SPACING_LUT` environmental variable. In memory are stored all the multiplications between operands representable in a word of length equal to `INPUT_RANGE` in two's complement. In other words they can range

from $-2^{\texttt{INPUT\_RANGE}-1}$ to $2^{\texttt{INPUT\_RANGE}-1} - 1$. The incremental step between two consecutive operands is $2^{\texttt{SPACING\_LUT}}$.
Taking as an example `INPUT_RANGE` = 4 and `SPACING_LUT` = 2 the results stored would look like:

$$-8 \times -8,\ -8 \times -4,\ \dots\ -8 \times 4,\ -4 \times -8,\ -4 \times -4,\ \dots\ 4 \times 0,\ 4 \times 4$$

As a consequence of this the length of the memory word must be twice he value of `INPUT_RANGE` to be able to store the whole result of the multiplication. With those parameters the platform can span from a dense LUT using
`SPACING_LUT` = 0 to an empty one, entirely calculating the output, using
`SPACING_LUT` = `INPUT_RANGE`.
The number of stored results is equal to $2^{\texttt{INPUT\_RANGE}-\texttt{SPACING\_LUT}}$ which multiplied by `WORD_SIZE` gives the maximum total number of bits occupied by the LUT. The memory initialization is performed at the beginning of the simulation if an instance of interpolator is present in the architecture.

When the interpolation operation begins the operands are parsed to extrapolate the memory address to interrogate. The partial result is then fetched and the exact result is reconstructed through the interpolator logic.
The only operation performed inside the interpolator are register shifts and additions, keeping the introduced latency low. Energy and delay estimations are computed using one of the adder operational card, neglecting shifts consumptions. The former is exposed to the component output interface through the `energy` signal.

Memory

The memory is the central part of the platform and it's the module from which every operation starts. Its architecture presents one input and one output data bus. It is composed of one main thread waiting for a power on signal. Once the start up time is elapsed athe ready signal is raised to listen for instructions. As soon as one of the two operation signals, `read` and `write`, are set to an high value the proper instruction is performed. After the correct delay is elapsed the action is performed.

Table 3.6. Table summarizing configuration, input and output parameters of the interpolator module.

env. variables	`INPUT_RANGE`	Operands number of bits
	`SPACING_LUT`	Sparsity of the LUT
input signals	`operation`	Signal to start the operation of the module
	`input`	Input bus from memory
	`operand_a`	First operand bus from the input decoder
	`operand_b`	Second operand bus from the input decoder
output signals	`ready`	Signal for module availability
	`output`	Output data bus
	`addr`	Memory address bus
	`energy`	Total energy consumption of the module

In the end the ready signal is raised again and the module falls into an idle state until the next input.
At the beginning of the simulation the interpolator LUT section is initialized storing all the results in the expected format. The process is explained in detail in 3.2.2.

Energy consumption is updated on the `energy_consumption_total` signal together with `static_power_consumption`. In order to have a finer level of detail in the output estimations the various components of the energy consumption are also exported.
They are: `energy_consumption_cells`, `energy_consumption_interconnect` and `energy_consumption_manager`.
The former is the only one exploited at the current stage of the project, the others are handled but give no contribution. They are present for forward compatibility with future enhancements of the simulator.
Parameters for these estimations comes from the chosen operational card. Two of them are available at the moment, one for DRAM cell technology and one for 1T-1C Ferroelectric cell technology.

Table 3.7. Table summarizing configuration, input and output parameters of the memory module.

env. variables	`WORD_SIZE`	Bit number of memory word
	`MEMORY_SIZE`	Number of word present
	`INPUT_RANGE`	Input range bit for data
	`SPACING_LUT`	Sparsity of interpolator LUT
input signals	`on_off`	Power up signal
	`addr`	Address bus
	`read`	Signal to start read operation
	`write`	Signal to start write operation
	`data_in`	Input data bus
output signals	`ready`	Signal for module availability
	`data_out`	Output data bus
	`energy_consumption_total`	Total energy consumption
	`energy_consumption_cells`	Cells energy consumption
	`energy_consumption_interconnect`	Connections energy consumption
	`energy_consumption_manager`	Control logic energy consumption
	`static_power_consumption`	Static power consumption
operational card	`energy_read_0`	Energy to read a ’0’
	`energy_read_1`	Energy to read a ’1’
	`energy_write_0_to_0`	Energy to write a ’0’ on a ’0’
	`energy_write_0_to_1`	Energy to write a ’0’ on a ’1’
	`energy_write_1_to_0`	Energy to write a ’1’ on a ’0’
	`energy_write_1_to_1`	Energy to write a ’1’ on a ’1’
	`power_keep_0`	Power to keep a ’0’ stored
	`power_keep_1`	Power to keep a ’1’ stored
	`energy_restart`	Energy to restart
	`read_latency`	Latency for read operation
	`write_latency`	Latency for write operation
	`shutdown_latency`	Latency for shutdown
	`restart_latency`	Latency for restart
	`retention_time`	Retention time for DRAM

3.3 Simulation validation and results

In the process of validating and collecting feedback on the platform functionality different benchmarks have been prepared.
All the following simulations have been performed with low level parameters derived by literature [22, 23, 24].

Matrix multiplication benchmark The first algorithm ran on the simulation platform is a matrix multiplication benchmark.
Five-by-five matrices of 8-bits signed integers elements are used as input. Both NWB and WB configuration are tested to compare the architectures' performances. The parameters used in 1T-1C memory cells operational card are showed in table 3.8.

Table 3.8. Parameters used in 1T-1C memory cells operational card. They are derived by low level CAD simulations.

Parameter	Value
energy to read '0'	1 nJ
energy to read '1'	2.5 nJ
energy to write '0' over '0'	0.5 nJ
energy to write '0' over '1'	2 nJ
energy to write '1' over '0'	2 nJ
energy to write '1' over '1'	0.5 nJ
power to store '0'	0.1 mW
power to store '1'	0.1 mW
restart energy	5 nJ
read latency	20 ns
write latency	20 ns
shutdown latency	200 ns

In Fig. 3.4 is shown the total energy consumption of the platform for both configurations. A relatively small difference is visible and it's due to the output bus been used in the NWB operations to send back the result, increasing the total amount with respect to the counterpart.

It is important to specify that in the NWB benchmark, in order to compute the total error, results are stored back into memory for subsequent use. This means an additional write operation for each addition and multiplication is needed, increasing the total energy consumption.
This is definitely a consequence of the absence of a central computing unit inside the simulation. Having to handle all the data transformations by itself, the platform is forced to drift apart from the ideal benchmark. In further developments this would be a crucial enhancement.
In Fig. 3.5 is shown the distribution of relative error with respect to the expected output for the same benchmark for each matrix element. It is computed iterating the algorithm with different inputs each time and registering the various distances, depicted in the graph. It is clear from the picture that for 32-bits memory words the precision is sufficient for an exact computation, while for 16-bits the error is been kept inside acceptable boundaries from the implementation of a saturated arithmetic.

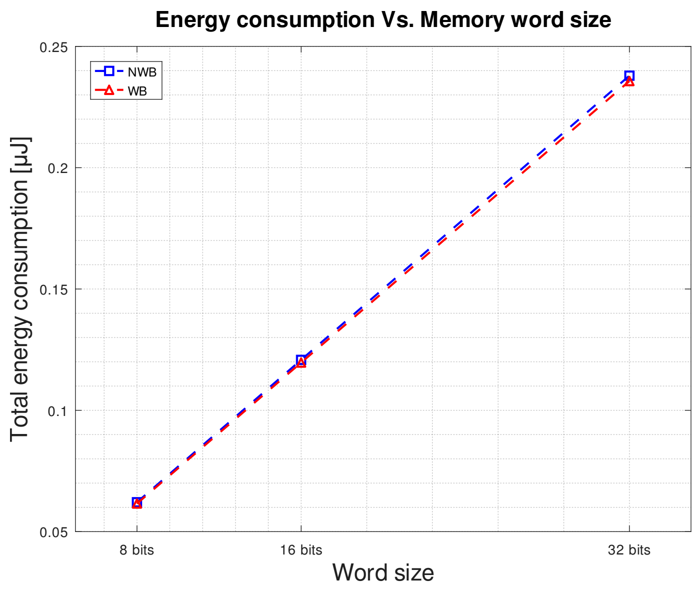

Figure 3.4. Platform energy consumption plotted against memory word size after the execution of matrix multiplication benchmark. Five-by-five matrices of random 8-bits signed integers elements used as input. In blue depicted the NWB architecture, in red WB architecture.

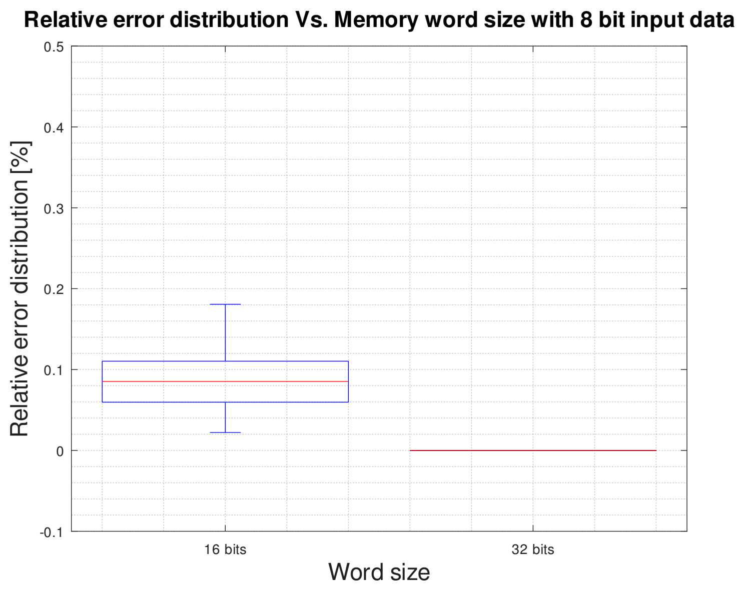

Figure 3.5. Output relative error distribution of matrix multiplication benchmark for 16 and 32 bits memory words.

Interpolator simulations Another way equally useful to exploit the simulator is to perform tests on the actual performances of emerging technologies nodes when placed into a full system. We can compare those to CMOS equivalents utilizing the same benchmark in order to understand if new designs are competitive. If that's not the case we can look for bottlenecks or weak points on which further research can focus on.
Extensive simulations were performed on the interpolator to discover its performances dependency when varying the physical parameters underneath.
The aim was to understand if the utilization of such a computing module was more advantageous with respect to using a more standard multiplier. In case of a negative response the next interest would it be how much the technology node would have to improve in order to accomplish that.

The module is been analyzed at first with the full range of 8-bits numbers and later with pseudo-random generated inputs. Those sets of operands are tested with different degree of LUT sparsity. This physically modify the dimensions of the memory matrix needed to store partial results, this is reflected on the amount of calculations required to retrieve the final one that decrease with a denser matrix.
For the memory cells a ferroelectric 1T-1C structure is been considered.
The trend depicted in Fig. 3.6 emerged from the simulations, where the average energy consumption per operation is plotted against the normalized area:

$$A = \frac{A_{\text{LUT}}}{A_{\text{cell}}}$$

The first point ($A = 0$) is used as reference since is the result obtained without a look-up table, therefore using just the logic operations like a normal multiplier.
By opposition the last value is the one with minimal sparsity, in other words utilizing a pure LUT approach.
It's clear from the graph that the current memory technological node, in blue, is not mature enough to compete in terms of energy with a dedicated hardware multiplier.
After a parametric sweep on the physical characteristics is been individuated a point in which this stops being the case.
With a 80% reduction in the energy required from the ferroelectric cells to perform

a reading operation we can see how the interpolator starts to be advantageous for a matrix larger than 1024 words. The upper bound of the sparsity imposed on the LUT can be defined by other means, for example an Energy-Area trade-off. This result is to be interpreted as a proof of the platform flexibility and as an example of integration of it in a wider workflow to help the development of emerging technologies both on a low, physical, level and an higher, architectural, one.

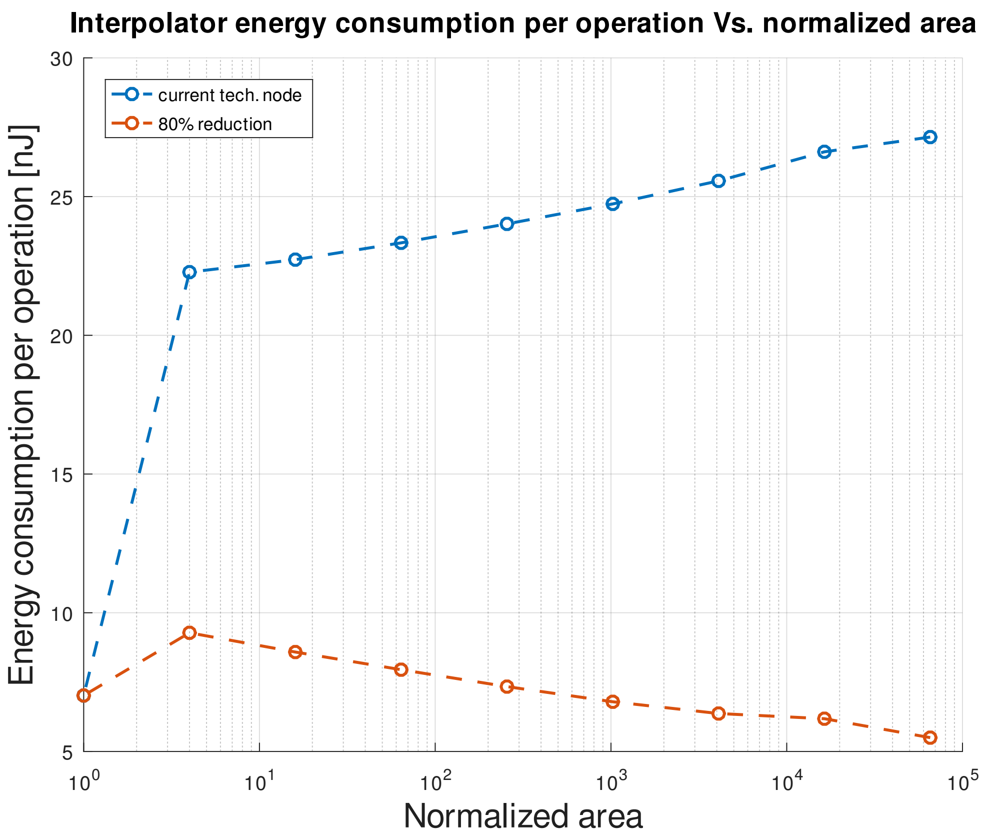

Figure 3.6. Energy consumption per operation vs. normalized area of the interpolator module where 8-bits pseudo-random signed integers used as input for the benchmark. In blue the actual ferroelectric 1T1C ferroelectric technological node, in orange an 80% reduction in read parameters, where an advantageous use of the LUT starts to emerge.

3.3.1 Further work

Results provided by the platform until this point are promising but they are far from optimal. There is a great margin from improvement in various areas of the system and given the generality of the tasks performed by it the path that could be walked are very different. Obviously the implementation of a computing unit inside the simulation would grant far more accurate estimations in terms of performances of the accelerator. In fact this would permit to use benchmarks that are closer to the real implementation, allowing the platform to stop reproducing operations that it should not execute.
Furthermore it would permit to take into account the energy saved by offloading computations alongside offering a traditional simulation for comparison.
Another side effect would be a more reliable simulation of energy consumption due to communications between the two modules.

The second area of enhancements it's in the benchmark phase. Actually few custom ones are performed but the simple interface created could be compatible with different inputs method. An interesting tool that could lower the barrier to simulations would be a modified compiler, aware of the LiM accelerator, to be able to automatically convert industry standard benchmarks into usable simulation inputs.

Finally, in the scope of the 3ϵFerro project, an integration into an automated design space exploration pipeline could be developed.

3.4 Conclusions

In this thesis the design of a simulation platform able to extract performance estimation across a wide range of LiM architectures is been presented. The simulator must be able to easily switch between different technologies as well, having a fine detail of modularity.
The main goal of this project is to provide a solid ground for its future development.

The actual state of the simulator is heavily focused on ferroelectric technology in order to be exploited in a design space exploration workflow in the context of the existing 3ϵFerro.

The results exposed into chapter 3.3 are not groundbreaking but they are proof of the extreme flexibility that this tool can provide in both comparing architectures and technologies between them.
In order to execute faithful high level estimations further development are needed, the most important one being the addition of a main computing unit to direct the accelerator operations.This would allow for more accurate benchmark to be executed, resulting in better estimations of the LiM structure.

The platform proved itself capable of giving useful insights on the technology used. This could be a game changer for establishing optimization pipelines on those parameters. Hardware could be developed having system level performances in mind, speeding up the overall workflow.
In this context a custom compiler able to delegate operations to the accelerator would be a great addition. Being able to use industry standard benchmarks would allow for an easier comparison of the results obtained with existing figures of merit, together with a wider spectrum of target applications.

Bibliography

[1] J. Valasek, “Piezo-Electric and Allied Phenomena in Rochelle Salt,” *Physical Review*, vol. 17, pp. 475–481, Apr. 1921.

[2] Littlewood P.B., “Physics of ferroelectrics,” Jan. 2002.

[3] M. Panjan, “Ferroelectrics and ferroelectric domains,” p. 17, Apr. 2003.

[4] P. D. Lomenzo, “Ferroelectric and antiferroelectric properties of hfo2-based thin films,” p. 209, 2016.

[5] T.S. Böscke, J. Müller, D. Bräuhaus, U. Schröder, and U. Böttger, “Ferroelectricity in hafnium oxide thin films,” *Applied Physics Letters*, vol. 99, p. 102903, Sept. 2011.

[6] J. Müller, T. S. Böscke, U. Schröder, S. Mueller, D. Bräuhaus, U. Böttger, L. Frey, and T. Mikolajick, “Ferroelectricity in Simple Binary ZrO2 and HfO2,” *Nano Letters*, vol. 12, pp. 4318–4323, Aug. 2012.

[7] Hans-Jürgen Butt, Karlheinz Graf, and Michael Kappl, *Physics and Chemistry of Interfaces*. third ed., Apr. 2013.

[8] A. Säynätjoki, “Atomic-layer-deposited thin films for silicon nanophotonics,” *SPIE Newsroom*, May 2012.

[9] O. Oluwatosin Abegunde, E. Titilayo Akinlabi, O. Philip Oladijo, S. Akinlabi, and A. Uchenna Ude, “Overview of thin film deposition techniques,” *AIMS Materials Science*, vol. 6, no. 2, pp. 174–199, 2019.

[10] H. Takasu, “The Ferroelectric Memory and its Applications,” p. 12, 2000.

[11] G. Moore, “Cramming More Components Onto Integrated Circuits,” *Proceedings of the IEEE*, vol. 86, pp. 82–85, Jan. 1998.

[12] J. Robertson and R. M. Wallace, “High-K materials and metal gates for CMOS applications,” *Materials Science and Engineering: R: Reports*, vol. 88, pp. 1–41, Feb. 2015.

[13] Q.-F. Ou, B.-S. Xiong, L. Yu, J. Wen, L. Wang, and Y. Tong, “In-Memory Logic Operations and Neuromorphic Computing in Non-Volatile Random Access Memory,” *Materials*, vol. 13, p. 3532, Aug. 2020.

[14] J. Junquera and P. Ghosez, “Critical thickness for ferroelectricity in perovskite ultrathin films,” *Nature*, vol. 422, pp. 506–509, Apr. 2003.

[15] W. Ahn, D. Jung, Y. Hong, H. Kim, Y. Kang, S. Kang, H. Kim, J.-H. Kim, W. Jung, J. Jung, H. Ko, D. Choi, S. Kim, E. Lee, J. Kang, C. Wei, S. Lee, K. A, and H. S. Jung, “A methodology to characterize device-level endurance in 1T1C (1-transistor and 1-capacitor) FRAM,” in *2008 17th IEEE International Symposium on the Applications of Ferroelectrics*, (Santa Re, NM, USA), pp. 1–4, IEEE, Feb. 2008.

[16] S. Dunkel, M. Trentzsch, R. Richter, P. Moll, and C. Fuchs, “A FeFET based super-low-power ultra-fast embedded NVM technology for 22nm FDSOI and beyond,” in *2017 IEEE International Electron Devices Meeting (IEDM)*, (San Francisco, CA, USA), pp. 19.7.1–19.7.4, IEEE, Dec. 2017.

[17] Y. A. Genenko, J. Glaum, M. J. Hoffmann, and K. Albe, “Mechanisms of aging and fatigue in ferroelectrics,” *Materials Science and Engineering: B*, vol. 192, pp. 52–82, Feb. 2015.

[18] I. O’Connor, L. Mozzone, M. Cantan, A. Bosio, D. Deleruyelle, and C. Marchand, “Granularity Exploration for Logic in Memory,” 2020. Publisher: Unpublished.

[19] I. O’Connor, M. Cantan, C. Marchand, B. Vilquin, S. Slesazeck, E. T. Breyer, H. Mulaosmanovic, T. Mikolajick, B. Giraud, J.-P. Noel, A. Ionescu, and I. Stolichnov, “Prospects for energy-efficient edge computing with integrated HfO2-based ferroelectric devices,” in *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, (Verona, Italy), pp. 180–183, IEEE, Oct. 2018.

[20] S. H. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, “NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads,” in *2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, (CA, USA), pp. 190–200, IEEE, Mar. 2014.

[21] X. Chen, M. Niemier, and X. S. Hu, “Nonvolatile Lookup Table Design Based on

Ferroelectric Field-Effect Transistors," in *2018 IEEE International Symposium on Circuits and Systems (ISCAS)*, (Florence), pp. 1–5, IEEE, 2018.

[22] M. Jhamb, Garima, and H. Lohani, "Design, implementation and performance comparison of multiplier topologies in power-delay space," *Engineering Science and Technology, an International Journal*, vol. 19, pp. 355–363, Mar. 2016.

[23] T. Vogelsang, "Understanding the Energy Consumption of Dynamic Random Access Memories," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, (Atlanta, GA, USA), pp. 363–374, IEEE, Dec. 2010.

[24] D. Sinha, K. Deepmala, T. Sharma, K. G. Sharma, M. Tech, and M. Tech, "New Design for Low Power High Performance 8T Full Adder," p. 4.

[25] J. Wang, H. P. Li, and R. Stevens, "Hafnia and hafnia-toughened ceramics," *Journal of Materials Science*, vol. 27, pp. 5397–5430, Oct. 1992.