POLITECNICO DI TORINO

Corso di Laurea Magistrale in Ingegneria Elettronica

Tesi di Laurea Magistrale

Ferroelectric FET–based circuits for Logic–in–Memory computing

Relatore
Prof. Mariagrazia Graziano

Correlatore:
Prof. Giovanna Turvani

Candidato
Danilo Franco

Luglio 2021

Contents

Chapter 1

Abstract

Conventional architectures for digital computing are based on the Von–Neumann paradigm, where data are exchanged between CPU and memory. That is, any program executed by the CPU requires a certain amount of data from the memory, in order to elaborate it and write back to the memory. Nonetheless, in modern computer technology, the tendency is that CPUs are faster than memories, causing the CPU to wait for data coming from the memory and resulting in a slow down of the processing speed in computers and other elaboration devices.
Therefore, solutions have been elaborated during the past decades. Among the fundamental ones: fractioning the memory unit allocating the faster units closer to the CPU, elaborating new memory topologies and lastly the Logic in Memory approach.
The Logic in Memory approach aims to execute inside the memory array part of the logic and arithmetic operations conventionally reserved to the ALU unit in the CPU.
Moreover, this new paradigm is being improved by the introduction of *beyond–CMOS* devices. In other words, an alternative to standard CMOS–based digital circuits is being researched in every technology which is capable of logic and storage functionalities. Then, the focus of this work is the Ferroelectric Field Effect Transistor (FeFET), an enanched MOSFET which shows hysteresis in its gate voltage–gate charge characteristic, as discussed in [32].
In this work, the FeFET is treated first under the mathematical point of view, by means of the Landau-Khalatnikov equations which predict the response of the device to the application of an external voltage. Then, a Verilog–A model for the FeFET is proposed and discussed at the schematic level in the

CAD software for microelectronics Cadence Virtuoso, simulated and characterized.
Furthermore, it is demonstred that the FeFET is suitable for both memory and logic goals. Nevertheless, before discussing FeFET–based Logic in Memory solutions, which is the final purpose if this work, memory arrays of three different dimensions are designed and simulated in Cadence Virtuoso, referring to the memory topologies discussed in [39]. Moreover, a sense amplifier peripheral is developed. Figure 1.1 shows the memory cell simulated for this

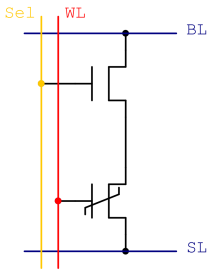
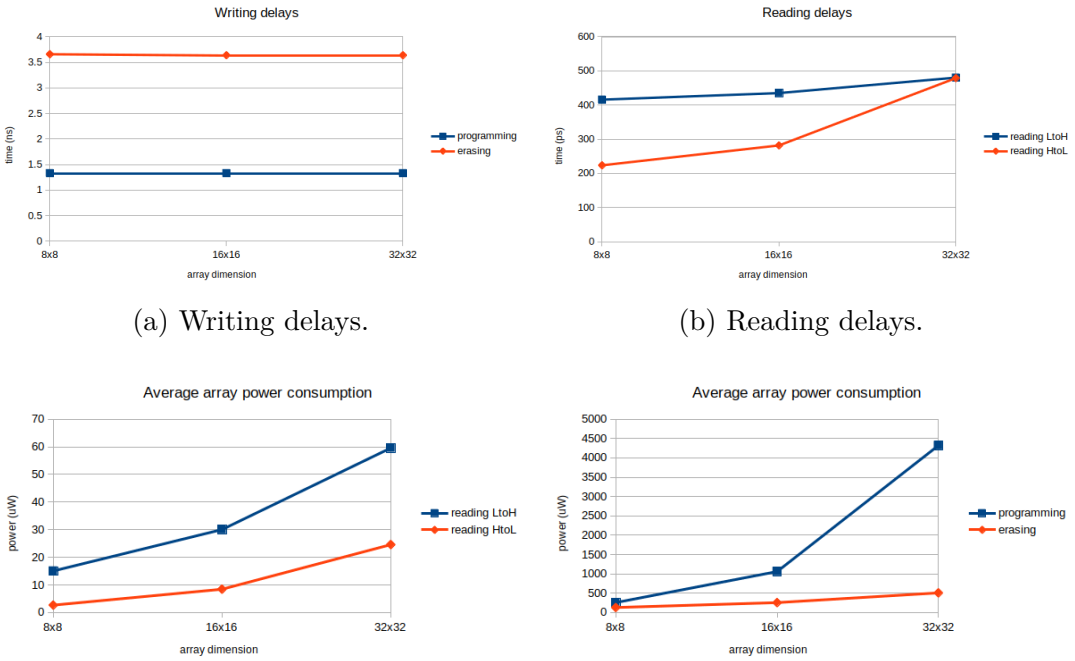


Figure 1.1: FeFET–based memory cell.

purpose.
The resultant characterization of a FeFET–based memory is presented in Figure 1.2, for the array dimension of 8, 16, 32 bit and 8, 16, 32 words. In general, writing '1' appears to be faster than writing '0', due to asymmetries in the FeFET hysteresis. Nonetheless, the dynamic power in writing '1' is higher due to the leakage of current.

For what concerns the developement of Logic in Memory, it is chosen to characterize single cells whose features are compared in Figure 1.3. The presented Logic in Memory cells are a selection of six cells capable of memory and logic. In particular, a Look Up Table cell is created, which can execute all the logic functions of two inputs. Moreover, some of them have a programmable function. For each cell, the schematic level and the correlated simulations are discussed.
Finally, the concept of Liberty description is introduced, which aims to define for a given logic gate or cell the performance in terms of timing and power. That is, these data are exploited by logic synthesis tools to select which gate best fits a given architecture.

The results of this characterization are showed in Figure 1.4. Here, for each measurement and each cell, a sample of data is randomly selected among one of the input–output combinations and then the worst value from all the available data is considered.



(a) Writing delays.

(b) Reading delays.

(c) Average dynamic reading power.

(d) Average dynamic writing power.

Figure 1.2: Results of the timing and power performance of the memory arrays.

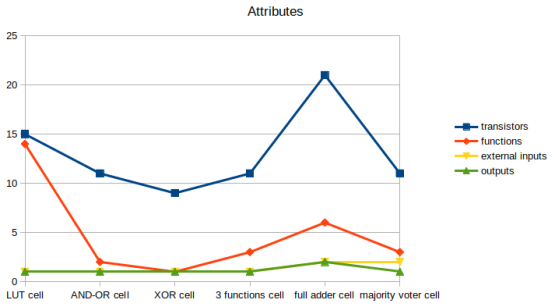
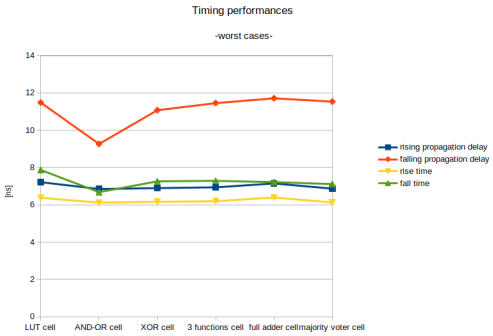
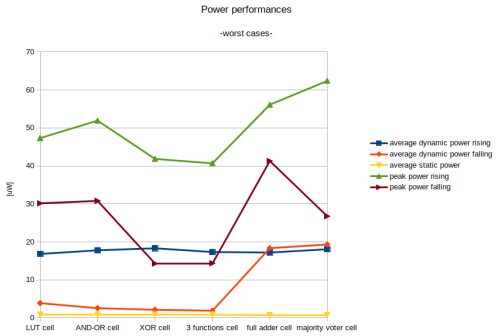


Figure 1.3: Prospect of cell size, number of functions, number of inputs and number of outputs from the different LiM cells.

Regarding the timing performance, by comparison of the results discussed, the delays which affect the output of the logic functions are almost one order of magnitude greater than the reading and writing delays of the conventional array. Morevoer, the average dynamic power dissipated by the Logic in Memory cells is comparable to the writing power dissipated by a cell in a standard



(a) Prospect of the timing measurements from the different LiM cells.



(b) Prospect of the power measurements from the different LiM cells.

Figure 1.4: Liberty characterization of the Logic in Memory cells.

array, but an order of magnitude greater than its reading power.
In conclusion, the logic functionality introduced to the FeFET–based memory is time and power demanding with respect to the operation allowed in a standard memory. Nonetheless, these enhanced cells can be used to build a Logic in Memory array, avoiding the whole memory content transferring to the CPU, as discussed before.

Chapter 2

State of the art

Overview

Over the past 50 years progress in computing and information technology, as well as the business of the semiconductor industry, has been based on the downscaling of the MOSFET transistor and on its derivative Complementary MOS (CMOS) digital systems.
As a matter of fact, CMOS–based computing systems sustained an exponential increase in both the operating frequency and the area density until, nowadays, at least two technological barriers have risen, resulting in limiting these performance.
That is: the first is the dissipated power having become so large, the second is the increasing speed gap between the central unit and the memory unit, problem known as *memory wall* or *bottleneck*, leading to a cap to the operating frequency. This aspect is explored in section 2.1.
Therefore, the demand for higher performance is expected to be no longer satisfied by the conventional solutions in circuit design that the standard CMOS has provided so far.

2.1 Logic–in–Memory

2.1.1 Von Neumman architecture

Conventional computer architectures are designed according to the **Von Neumann** paradigm. That is, there are three separate units, respectively for computing (CPU), for storage of data and instructions (memory), and

for data–exchange (bus).
In this context, while the CPU has reached its full potential, the memory stays behind. Indeed, the response time of a modern memory interfacing with a modern CPU has become the system bottleneck.
Modern studies are tackling down these barriers from many angles, from the physical level to the architectural one. A wide exploration of new paradigms is presented in [1].
Moreover, some of these studies suggest a massive use of parallelism, such as Graphics Processing Unit (GPU). These devices make extensive use of multi–core processing and each core often gets a dedicated high–throughput connection with the memory ([4]).
Besides, application-specific processors known as accelerators are integrated in the computing systems to speed up a set of algorithms to be performed ([34], [35]).
Furthermore, memory chips with enhanced bandwidth have been investigated, such as the hybrid memory cube (HMC, [36]) and the high bandwidth memory (HBM, [37]), where performance are improved by stacking multiple memory chips in a 3D structure.

2.1.2 LiM with standard CMOS technology

A thriving research field which goes beyond the approaches previously expressed exists, which is suggesting the approach of *Logic–in–Memory* (LiM) as a solution for the aforementioned memory issues. That is a newly developed architectural concept which tries to overcome the separation between logic and memory units imposed until now. A good overview of this research is given in [2], [3].
As the name suggests, computation is partially or totally moved inside the memory array bringing benefits such as reduction of power consumption originated by data movement between central and memory units, avoiding of the bandwidth bottleneck and distributed computation.
The basic implementation of LiM allows the integration of logic elements in every memory cell.

A futher evolution of the *LiM* architecture is named *Configurable Logic in Memory Architecture* (CLiMA), conceived to allow flexible and adaptable design. CLiMA is well explained in [1].
The main point of a CLiMA is that the designer can choose how to map each and every operation of an algorithm thanks to logic units moved both inside

the single cell and to the pheriperal of the memory array.
In fact, a CLiMA schematic is depicted in Figure 2.1, where it is clearly represented how a cell is the result of merged storage and logic features, and then put in an array with logic pheripherals. This is the most general and straightforward approach, which allows to obtain the desired algorithm inside the memory.

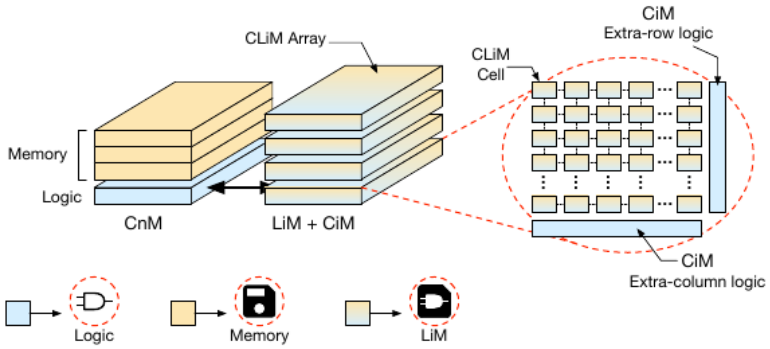


Figure 2.1: CLiMA in-memory computing approach [1].

By definition, an enhanced cell with built–in logic is called *smart cell*. It is made by a storage element implemented in one of the known technologies (such as Static Random Access Memory, Dinamic RAM, or more modern Resistive RAM, Magnetic RAM) and the logic element capable of executing the most common bit–wise operations.
Usually, a LiM or a CLiM architecture is designed to perform different algorithms, starting from the bit–wise computation inside the cell and arriving to more complex inter–row computations, involving more than one row in the memory array.

As far as performance of such these architectures are concerned, recent investigations are directioned toward new technologies to replace CMOS or to be integrated with it. According to the fact that standard memory arrays, based on alternative technologies, had been already analysed so far, the next step consist of building up entire smart cells exploiting the opportunities that non–volatile beyond–CMOS devices might offer.

2.2 Beyond–CMOS technologies

2.2.1 Memory devices

New and emerging non–volatile memory concepts have been introduced into the traditional memory hierarchy.
These, generally identified under the name of resistive and capacitive switching devices, do not base their functionality upon charge, as in traditional metal oxide semiconductor transistor technology, but instead on different physical characteristics of the active material they are made of. And even if they rely on charge, as the ferroelectric capacitors and ferroelectric transistors, they distinguish themselves from the conventional semiconductor devices by the way they are employeed, as it will be explained in this section.
Among these, the most developed and already available on the market are the resistance switching memory (RRAM), phase change memory (PCM), magnetoresistive RAM (MRAM) and ferroeletric RAM (FeRAM) ([4]).

What is remarkable about these devices is their natural ability of merging both memory and computation within themselves, often through peculiar architecture designs such that LiM is achieved.

Threshold Switching Memristor

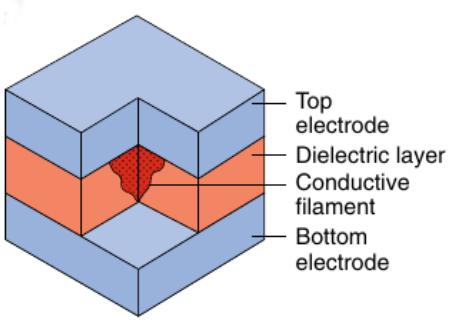


Figure 2.2: Resistive switching device [4].

Memristors are two-terminal resistors ([4], [5]) that modify the internal resistance state according to the history of applied voltage (or current). They are usually referred to as *memory resistors* in the sense that their resistance

state is used to memorize information. In other words, memristors are passive circuit components with their resistance state which recalls a pinched–hysteresis loop.
The first aim memristors were addressed to was the resistive random access memory (ReRAM), but their non–linear switching property suggests logic and computational implementations.

A common memristor consist of a metal–insulator–metal stack where a filamentary path is initially induced, as in Figure 2.2. At this point, the application of a positive voltage allows the defects migrate inside the insulator causing the transition to the low–resistance state (LRS). On the contrary, the application of a negative voltage forces the disconnection of the conductive filament and thus an high–resistance state (HRS).
These transitions are visible in Figure 2.3, where the non-volatile storage ability of this device is observable. That is, the resistance status is not lost when the device is not driven by any current or voltage.

For what concern the advantages, these memories can be accomodated in a crosspoint structure as a result of not being three–terminal devices, and each of them is independently programmable and erasable. Prototypes of such architecture have been also presented in the form of a one transistor/one memristor cell, as in [38].
Furthermore, they provide fast switching and moderate endurance compared to conventional memory arrays. A summary is presented in Table 2.1.

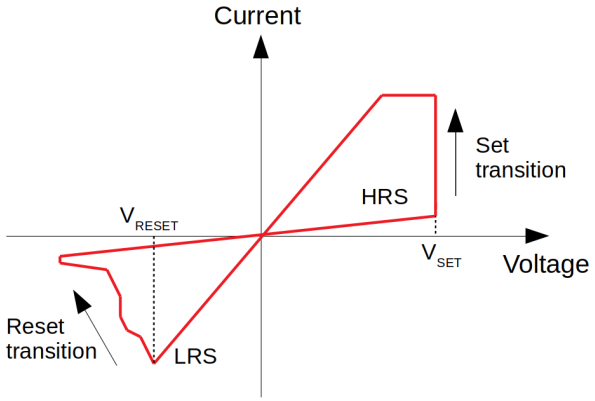


Figure 2.3: Current-voltage characteristic of a memristor [4].

	ReRAM	**SRAM**	**DRAM**	**NAND Flash**
Cell area	$< 4\,F^2$	$> 100\,F^2$	$6\,F^2$	$< 4\,F^2$
Read time	< 10 ns	~ 1 ns	~ 10 ns	~ 10 µs
Write time	< 10 ns	~ 1 ns	~ 10 ns	100 µs-1 ms
Write energy [J/bit]	~ 0.1 pJ	~ 1 fJ	~ 10 fJ	~ 10 fJ
Endurance	$\sim 10^6$-10^{12}	$> 10^{16}$	$> 10^{16}$	$> 10^4$

Table 2.1: Performance review of different memories technologies [30].

Phase Change Memory

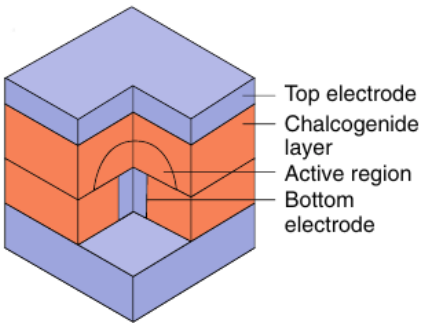


Figure 2.4: Phase change memory cell structure. [4]

Every material that exists in at least two structurally distinct solid phases, amorphous and crystalline, is defined as a *phase change material* (PCM, [7]). Amorphous and crystalline phases clearly show different optical and electrical properties, therefore such materials are being used to store information in electronic applications, as long as the resistance state in the amorphous phase is higher than in the crystalized phase.

The structure of a basic PCM cell ([4]) is shown in Figure 2.4, a two terminal device which has the typical mushroom shape. Among the two electrodes, there is a chalcogenide active layer, such as $Ge_2Sb_2Te_5$, which gets heated and, so, physically rearranged by the application of voltage pulses.
In particular, the amount of crystalline volume in the active layer is augmented by applying long enough and low amplitude pulses, while fast and

high amplitude pulses lead to local melting and consequent amorphization. This operation principle is also depicted in Figure 2.5.

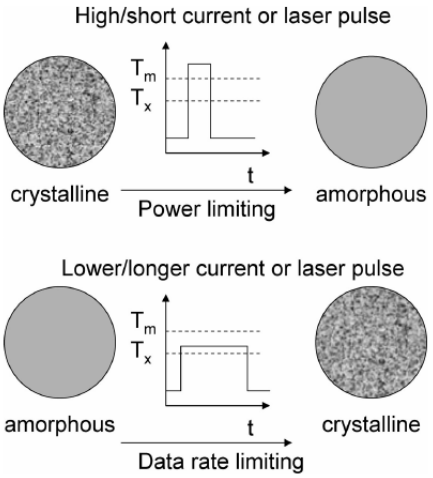


Figure 2.5: PCM resistance change process [7].

Moreover, two main applications of PCM exist nowadays: rewritable optical phase change storage technology and the emerging PCM Random Access Memory (PCMRAM), as explained in [7].
Finally, PCMs have been described in this section but will be no longer considered in this work. The reason is that delicate operations such as crystallization and melting are found to be, respectively, data rate and power limiting [7]. Indeed, the melting temperature for phase change materials is typically between 500 and 800 °C. That is, the performances of PCM–based memory arrays are discouraging, as summarized in Table 2.2. Moreover, the most common usage of this technology is optical, as well.

	PCM	**ReRAM**
Cell area	4–20 F^2	4 F^2
Read time	< 10 ns	< 10 ns
Write time	~ 50 ns	< 10 ns
Write energy [J/bit]	~ 10 pJ	~ 0.1 pJ
Endurance	$> 10^9$	$\sim 10^6$-10^{12}

Table 2.2: Performance comparison between PCM-based memories and memristor-based memories [30].

Magneto Tunnel Junction

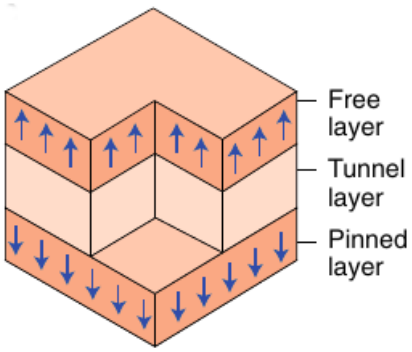


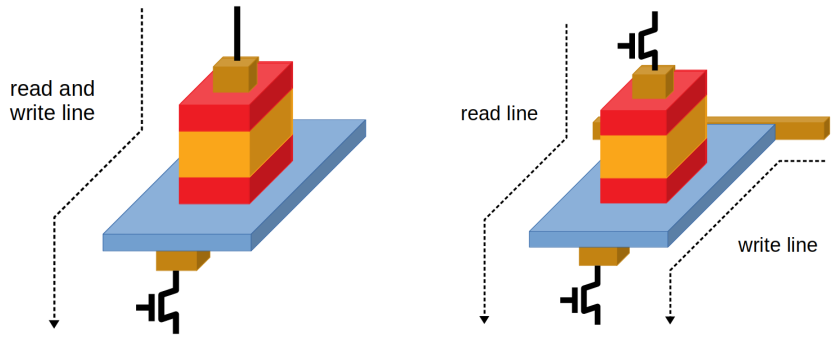
Figure 2.6: Magneto Tunnel Junction cell structure [4].

A *Magneto Tunnel Junction* (MTJ), in Figure 2.6, consists of a MIM structure with two ferromagnetic metal layers, usually CoFeB, and a thin oxide, usually MgO ([4], [17]).
One of the two metal layers is defined as *pinned*, meaning that its ferromagnetic magnetization is structurally fixed to act as a reference, while the magnetization of the other layer is controllable. The latter is called *free* layer.
Consequently, an MTJ device can either be in the state of low resistance, when the ferromagnetic polarizations are parallel, or high resistance, when these are anti–parallel.
Actually, there are two well known techniques used to flip the state of the MTJ, that is the *Spin Transfer Torque* (STT) and the *Spin Orbit Torque* (SOT). Besides, they also give the name to the memories, respectively STT–MRAM and SOT–MRAM. Treatments of this can be found in [9], [10], [11].

As far as the physical aspect is concerned, in a STT–MRAM cell the transition is conducted by spin–polarized electrons which rotate the free layer magnetic polarization by magnetic momentum conservation. That is to say, current is applied in a precise direction and the polarization is estabilished accordingly. As a result, the cell is a two-terminal component. STT-MRAM is known to have magnetorersistance ratio of about 200%, high switching speed (< 1 ns) and high endurance. [4]
Instead, the SOT–MRAM cell is made by a MTJ laid above a heavy metal film, which results in a three-terminal device as a matter of fact.
In this case, the flipping mechanism is due to the injection af a spin–polarized

current in the metal film, whose spin desity induces a spin orbit coupling to the free layer. The structures of these cells are showed in Figure 2.7.

Therefore, this cell is being introduced as an improvement of the STT–



(a) STT-MRAM cell. (b) SOT-MRAM cell.

Figure 2.7: Arrangments of MTJ cells [8].

MRAM cell, as long as the read and write paths are physically separated. Futhermore, SOT–MRAM offers better performance in terms of speed and endurance at the expense of a degraded density [8], as it is reported in Table 2.3 from [31].

	STT-MRAM for SRAM	**STT-MRAM for eFlash**	**SOT-MRAM**
Cell area	70-100 F^2	50-60 F^2	160 F^2
Read time	$\sim$ 5 ns	$\sim$ 25 ns	$\sim$ 5 ns
Write time	$\sim$ 10 ns	$\sim$ 200 ns	$<$ 2 ns
Endurance	10^{14}	10^{8}	10^{14}

Table 2.3: Performance comparison between STT-MRAM and SOT-MRAM. [31]

Ferroelectric capacitor

A *ferroelectric capacitor* (Fe–cap) recalls the structure of a regular capacitor except for the substitution of the dielectric with a ferroelectric material as shown in Figure 2.8, mostly perovskite material or doped–HfO_2 ([18], [19]). In particular, when a voltage is applied across its two terminals, the polarization charge in the Fe-cap shows a hysteresis loop characteristic (Figure 2.9)

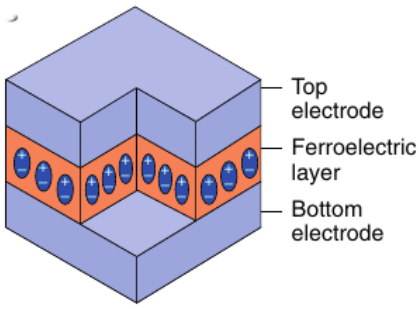


Figure 2.8: Ferroelectric capacitor cell structure [4].

due to the fact that the ferroelectric dipoles change their orientation. That is, its behavior is not different from the hysteresis of a ferromagnetic material, on which magnetic memory devices are based.

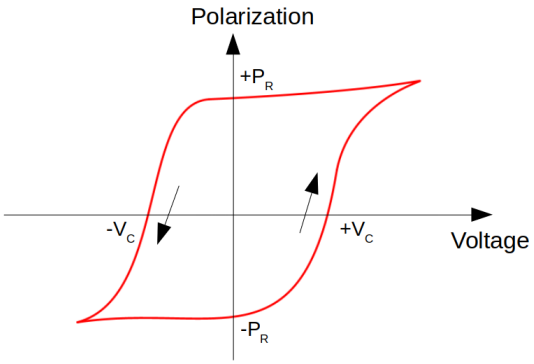


Figure 2.9: Hysteresis of a ferroelectric capacitor [4].

Therefore, even when driven by 0 V, the Fe–cap is able to retain a remnant-polarization charge, thus it is effectively a non–volatile memory element.
The Fe–cap cell is capable of reading and writing operations, by selecting and adoperating the correct voltage level. In fact, if the voltage is higher (in absolute value) than a specific threshold, called *coercitive* and set by technology, the polarization is switched.
Otherwise, if the reading voltage is lower than the coercive threshold, the remnant charge is theoretically not switched, nonetheless it may happen that the continuous application of pulses disturbs the stored non–volatile charge, as shown in Figure 2.10.
Moreover, the transitive speed of this cell is expected to be comparable to

DRAMs ([19]).

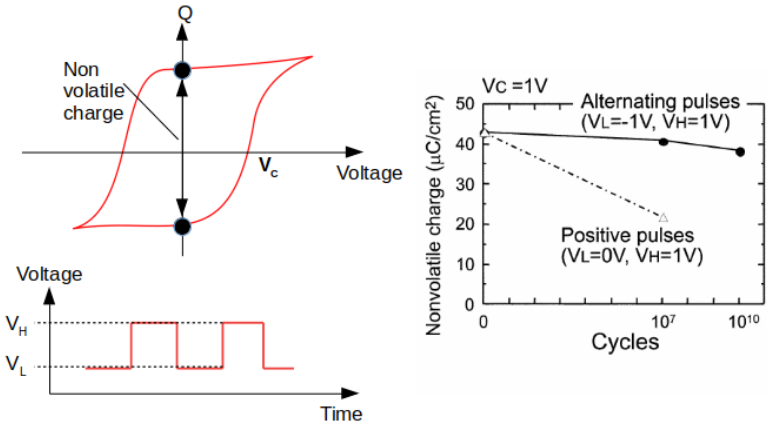
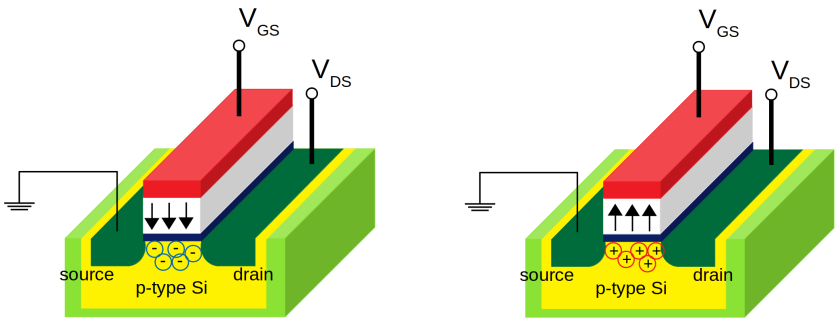


Figure 2.10: Non-volatile charge in a ferroelectric capacitor [12].

It is important to underline that the MIM stack resistance is not impacted in this way of use, but the charge induced on the metallic electrodes is. Nevertheless, resistance change is still achievable from the ferroelectric switching by the *ferroelectric field effect transistor* (FeFET), a three-terminal structure ([23]).
Here, the alternation in dielectric polarization causes a variation in the resistance of the FeFET channel.

Ferroelectric Field Effect Transistor



(a) Low threshold state. (b) High threshold state.

Figure 2.11: Structure of an n–type FeFET [25].

A *Ferroelectric FET* (FeFET) has the same structure as a standard MOS transistor except to the presence of a ferroelectric layer inside the gate stack (Figure 2.11).
A typical implementation of a FeFET features a TiN/Si:HfO_2/SiO/Si gate, and it is usually realized in two different technologies, the 22 nm FD-SOI and the 28 nm HKGM, as explored in [22].
Recalling the description of the hysteresis polarization of the ferroelectric capacitor, the application of either a large enough positive or negative gate voltage switches the polarization state of the ferroelectric gate. That is, negative or positive charges are inducted in the transistor channel, thus setting the devices into, respectively, the *low threshold* state or the *high threshold* state. These phenomena are usually referred respectively as *programming* and *erasing*, and are visible in the output characteristic in Figure 2.12. Here, the programmable thresholds of the FeFET are shown.

As far as memory and computing are concerned, the threshold–changing

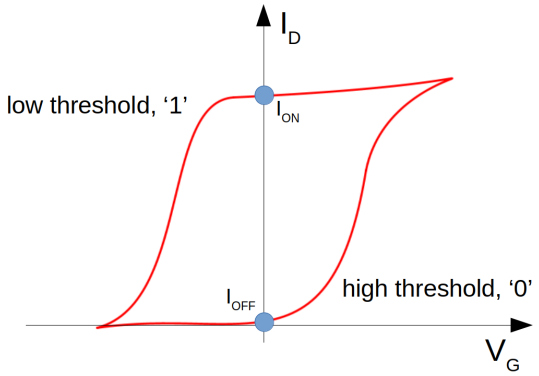


Figure 2.12: Hysteresis of a Ferroelectric transistor [22].

feature allows to store information, since a FeFET in high threshold state is not conducting (logic '0', transistor off) whereas it is conducting in low threshold state (logic '1', transistor on) even when $V_G \approx 0\,\mathrm{V}$. Conduction occurs for $V_{DS} > 0\,\mathrm{V}$.
Moreover, in the experimental cases expressed in [22], the combination of the polarization state of the transistor and the input gate voltage leads to the set up of logic boolean functions, that will be explained in detail in the next section.
Regarding other applications, a FeFET belongs to the field effect transistor family, thus it can be employed as a standard transitor. Such a device, which exists already in literature as *negative capacitance* FET (NCFET), is based

on the removal of the hysteresis behaviour from the output characteristic, in which the curve exhibits a better steepness than in standard CMOS transistors, being ideal for digital applications. An exhaustive theoretical digression about the NCFET is avaible in the first section of the manual in [29].

2.2.2 Smart cells and logic

In the past decades the technologies previously described in subsection 2.2.1 have been employed for the realization of either logic gates with non–volatile storage capabilities or non–volatile memory cells with logic capability.
Therefore such architectures have been demonstrated both experimentally and by means of computational models, as it is will be presented in this section, where a look–up at the state of the art has been made.

Resistive threshold logic

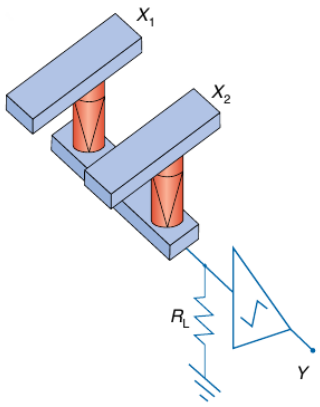


Figure 2.13: Resistive threshold logic. It is composed by two memristors (red) driven by two electrodes [4].

A resistive threshold logic is classificable as memristor–based logic gate. In fact, it represents a voltage–to–voltage boolean function like any standard logic port and its behavior is based on memristors and their resistive divider. Then, a threshold comparator drives the output to the logic level. The equivalent circuit, shown in Figure 2.13, is derived from [4].
Such a device has no storage capability and can have N inputs, as long as N input memristors are employed. Nonetheless, it can be exploited as the logic element in a LiM smart cell.

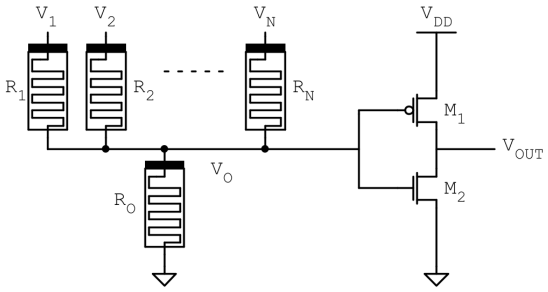


Figure 2.14: Resistive threshold logic with inverter-made comparator [13].

The typical logic functions programmable on this device are NAND and NOR. To explain the working principle, Figure 2.14 is taken as a reference. Here, an inverter is used as threshold comparator.
Then, in the case that the input memristors have the same value and the output resistor (memristor) is an integer multiple of the formers, the voltage divider is given by:

$$V_0 = \sum_{i=1}^{N} V_i/[1/m + N] \tag{2.1}$$

$$m = R_0/R_i \tag{2.2}$$

Thus, the number of inputs can be generic and the voltage levels for high and low digital values are arbitrarily chosen. For example, a standard dynamic for integrated circuit can be assumed, as $V_H = 1\,\mathrm{V},\ V_L = 0\,\mathrm{V}$.
Knowing that NAND function has output '1' whenever at least one input is '0' and NOR function has output '0' whenever at least one input is '1', the following boundary conditions are needed:

- for NAND:

$$(N-1)(1/m+N)^{-1} < V_{TH} < N(1/m+N)^{-1} \tag{2.3}$$

- for NOR:

$$0V < V_{TH} < (1/m+N)^{-1} \tag{2.4}$$

  For $V_H = 1\,\mathrm{V}$, $V_L = 0\,\mathrm{V}$. Also, these equations are valid for whatever threshold comparator is appended to the circuit.

Therefore the two design parameters for the circuit are V_{TH} and m, which are dependent on each other at a given N. If the case in Figure 2.14 is considered, $V_{TH} \approx 0.5\,\mathrm{V}$, since it corresponds to the inverter threshold voltage. N is so assumed as the sweep variable and the results are shown in Table 2.4.

Analyzing Table 2.4, for large N, the needed value of m in order to obtain

N	$V_{TH} \approx 0.5\,\mathrm{V}$
2	$0.5 < m$
3	$0.33 < m < 1$
4	$0.25 < m < 0.5$
20	$0.05 < m < 0.055$

Table 2.4: Resistive threshold logic parameters for NAND; V_{TH} fixed.

a NAND function is shrunk to a small range.
Instead, as far as NOR is concerned, no solution exist for Equation 2.4 if V_{TH} is fixed, because m would be negative.
Nevertheless, if the case that m is fixed is assumed and the threshold comparator is arbitrary –an operational amplifier or a rectifier– the results are shown in Table 2.5.

Also for this second approach, the parameters have to be accurate when N

N	V_{TH}, $m = 1$ [V]
2	$0.33 < V_{TH} < 0.66$
3	$0.5 < V_{TH} < 0.75$
4	$0.6 < V_{TH} < 0.8$
20	$0.905 < V_{TH} < 0.952$

(a) Results for NAND.

N	V_{TH}, $m = 4$ [V]
2	$V_{TH} < 0.44$
3	$V_{TH} < 0.307$
4	$V_{TH} < 0.190$
20	$V_{TH} < 0.049$

(b) Results for NOR.

Table 2.5: Resistive threshold logic parameters for NAND and NOR; m fixed.

is large.

Consequently, appropriate architectural solutions should be found. Among the most popular there are:

- Inverter buffer, Figure 2.15

For low N NOR applications, inverters are cascaded with increasing power supply V_{DD} such that the final output stage reaches the same dynamic as the input;

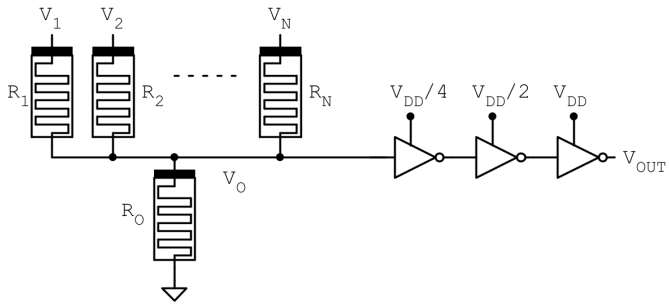


Figure 2.15: Resistive threshold logic with inverter buffer [13].

- Operational amplifier, Figure 2.16
  For high N applications, opamps are used as comparators with an arbitrary threshold. Thus, voltage levels very close to the threshold are sensed correctly.
  An operational amplifier–based resistive threshold logic has the advantages of being programmable and of providing a strong input–output decoupling. On the other hand, on Opamp requires at least eight transistors ([13]).

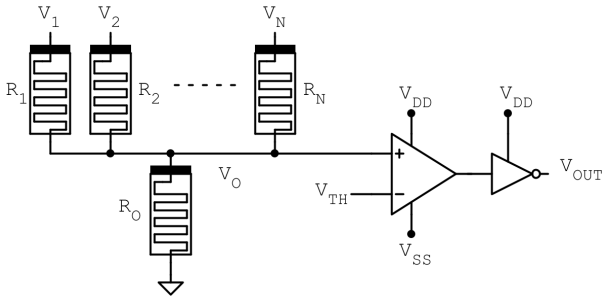
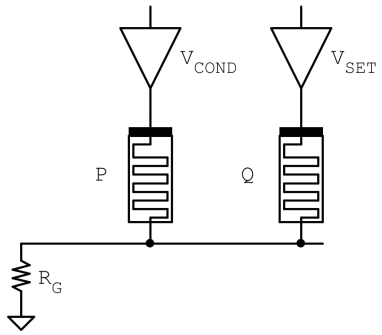


Figure 2.16: Resistive threshold logic with operational amplifier [13].

Stateful logic via material implication (IMPLY)

The material implication is a fundamental Boolean operation on two variables p and q such that $p \rightarrow q$ is equivalent to $\overline{p} + q$. Moreover, it is shown



(a) Schematic of the imply circuit.[6]

p	q	$p \rightarrow q$
0	0	1
0	1	1
1	0	0
1	1	1

(b) Logic implication truth table.

Figure 2.17: Schematic and truth table of the imply circuit.

that any Boolean function is feasible by means of a series of logic implications.
Remarkably, memristors can perform *stateful* logic operations by exploiting the implication, since they can act as *switches*. That is to say, if incorporated within an appropriate circuit, memristors serve simultaneously as gates and latches. The present section demonstrates these assumptions, referring to [6].
The schematic is shown in Figure 2.17a. Here, memristors P and Q represent the input values through their resistance state, which is for semplicity associated to open circuit (high) and closed circuit (low).
The operation is *destructive* because the output is written back to Q. Furthermore, this circuit can be integrated in a smart cell, where P is the memory element.

Moreover, to understand how this circuit works, it has to be considered that the applied voltages are chosen as follows:

$$|V_{COND}| < |V_{CLOSE}| \tag{2.5}$$
$$|V_{SET}| > |V_{CLOSE}| \tag{2.6}$$

where V_{CLOSE} is the low resistance threshold of the memristor, and

$$R_{closed} < R_G < R_{open}. \tag{2.7}$$

Therefore:

- when $P =$ '0', memristor P is excluded from the resistive divider and if $Q =$ '0', V_{SET} drops on it and closes it. Otherwise, Q stays closed;

- when $P =$ '1', most of V_{COND} drops on R_G, thus the voltage drop on Q is $V_{SET} - V_{COND}$. Q is then unchanged.

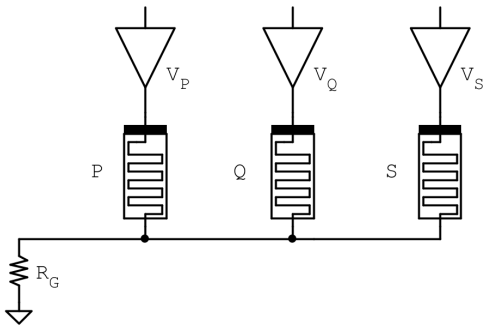


Figure 2.18: Schematic of NAND/XOR through imply circuits.

In addition, Figure 2.18 shows the schematic of a NAND function obtained according to the implication paradigm, from [6]. Here, three memristor are employed: P, Q as inputs and S as functional memristor. That is, the latter is used to store the temporary results as well as the final output.
Recalling that:

$$\overline{p \cdot q} = q \rightarrow \overline{p} \tag{2.8}$$

the operation is achieved in three steps

1. S is cleared (forced to '0') by applying V_{RESET};
2. $P \rightarrow S$ is stored into S, which corresponds to $S = \overline{p}$. In this case, V_{COND} is applied to P and V_{SET} to S;
3. $Q \rightarrow S$ is stored into S, which corresponds to $q \rightarrow \overline{p}$ and thus $\overline{p \cdot q}$. In this case, V_{COND} is applied to Q and V_{SET} to S.

In conclusion, the IMPLY logic is a versatile architectural solution, considering the fact that every boolean function can be achieved. Moreover, the circuit in Figure 2.18 is *non-destructive* for the inputs such that it can be integrated in a memristor smart array.
However, the latency increases with the function complexity, in the form of cascaded operations.
Finally the XOR operation in IMPLY logic is analyzed in the following.
First of all, Figure 2.18 shows the schematic for the current purpose, in which a total of three memristor are used and it is the same as the NAND case. P and Q are inputs, S is the functional memristor. However, for this function

the inputs are destroyed and the final output is stored in Q.
Assuming the De Morgan theorem for the boolean logic and the laws:

$$p \rightarrow q \Leftrightarrow \overline{p} + q = \overline{p \cdot \overline{q}} \tag{2.9}$$

$$r \rightarrow \text{`0'} \Leftrightarrow \overline{r} \tag{2.10}$$

it follows:

$$p \oplus q = (p \rightarrow q) \rightarrow ((q \rightarrow p) \rightarrow \text{`0'}) \tag{2.11}$$

Therefore, Equation 2.11 suggests that at least four distinct logic operations are needed to achieve the XOR function by means of the IMPLY logic. Actually, a destructive XOR operation is obtained in six steps:

1. Q is copied to S by a reading phase followed by a writing phase. Actually, the pheriperal architecture can be arranged to perform these operation in the same clock cycle;

2. $P \rightarrow S$ is stored into S, which corresponds to $p \rightarrow q$. In this case, V_{COND} is applied to P and V_{SET} to S;

3. $Q \rightarrow P$ is stored into P. In this case, V_{COND} is applied to Q and V_{SET} to P;

4. Q is cleared (forced to '0') by applying V_{RESET};

5. $P \rightarrow Q$ is stored into Q, which corresponds to $(q \rightarrow p) \rightarrow$ '0'. In this case, V_{COND} is applied to P and V_{SET} to Q;

6. $S \rightarrow Q$ is stored into Q, which corresponds to Equation 2.11. In this case, V_{COND} is applied to S and V_{SET} to Q.

Table 2.6 provides an overview of this process.
Finally, Figure 2.19 shows how the complexity of the logic function affects parameters such as latency and variety of voltage levels to manage.

Memristor Aided loGIC (MAGIC)

Memristor Aided loGIC addresses the realization of logic within passive crossbar memory arrays by means of memristive cells, in [14] defined as *Memory Processing Unit* (MPU).
MAGIC can be seen as an improvement of the stateful imply logic described before since it supports more basic Boolean functions, does not require additional resistors and most importantly every operation is non-destructive,

p	q	$p \rightarrow q$	$q \rightarrow p$	$(q \rightarrow p) \rightarrow$ '0'	XOR
0	0	1	1	0	0
0	1	1	0	1	1
1	0	0	1	0	1
1	1	1	1	0	0

Table 2.6: XOR through logic implications truth table. Ref. Equation 2.11.

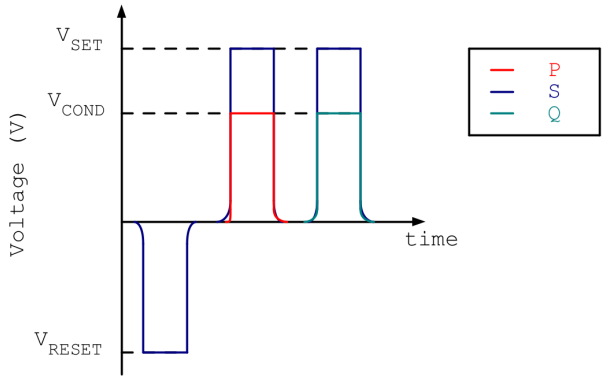
since the output cell is separated from the input cells.

The structure in Figure 2.20 is *defacto* a LiM architecture, where columns and rows are driven by voltage controllers allowing reading, writing and logic operations togheter with a battery of peripheral sense amplifiers for both rows and columns.
Intuitively, each cell stores a logic value according to its resistance state.
For what concern the computation, the *Voltage ThrEshold Adaptive Memristor* (VTEAM) model is adopted to explain the successively introduced equations, and can be found in [14], [15].

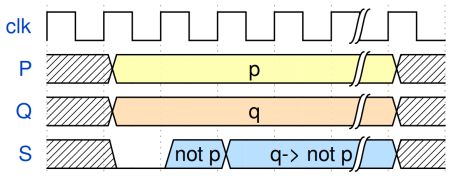
In the following, the three main operations available are explained.

- **Write**: since each column and row is individually driven, either V_{SET} or V_{RESET} can be applied to write, respectively, '1' or '0', as shown in Figure 2.21. Moreover, these voltages are chosen larger than each threshold voltage.
  Nevertheless, the write interference that may happen along the row and the column of the interested cell is attenuated by the application of half–voltages, as in Figure 2.21 and explained in [14]. That is, the adjacent rows and columns are driven by V_{ISO} (for instance $V_{SET}/2$ and $V_{RESET}/2$) in order to have a small voltage drop on the cells which surround the memristror to be written.

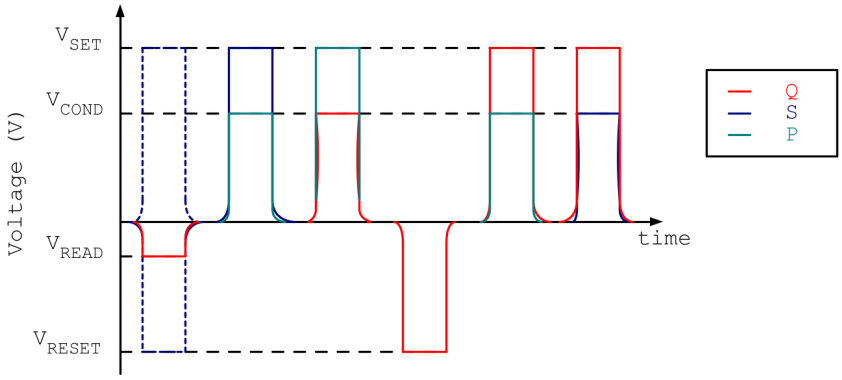
- **Read**: the read operation is performed by applying V_{READ}, below threshold, to the row (column) and the current is sensed by the sense amplifier attached.



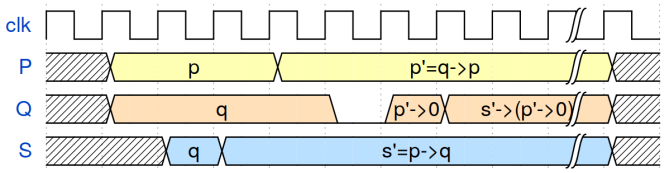
(a) Time graph of the NAND–imply logic, applied voltages.



(b) Time graph of the NAND–imply logic, digital values.



(c) Time graph of the XOR–imply logic, applied voltages.



(d) Time graph of the XOR–imply logic, digital values.

Figure 2.19: Time graphs of the imply logic.

- **NOR operation**: the most straightforward logic operation for this architecture is the NOR with N inputs.
  First of all, the operation involves only cells from the same row (column). Then, all the input cells are driven with the same voltage V_O (0 V) and the output cell with 0 V (V_O). Thus, the equivalent circuit is a resistive divider as depicted in Figure 2.22.
  After an initialization step where the output cell is set to '1' (or R_{ON}),

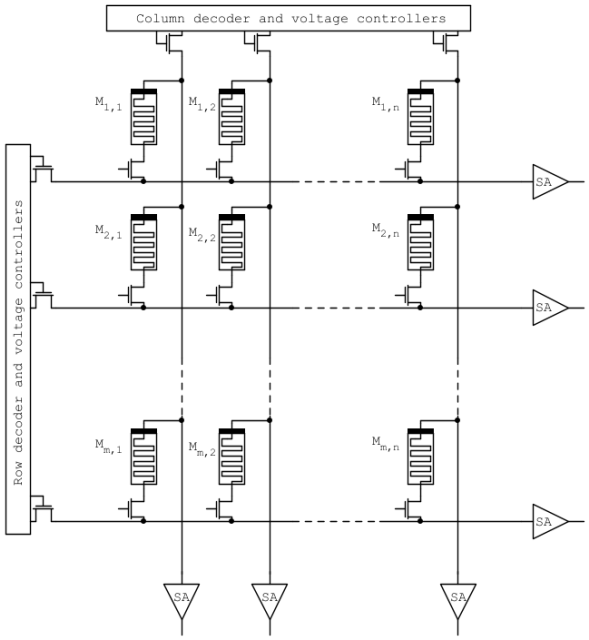


Figure 2.20: Schematic of the Magic crossbar architecture [14].

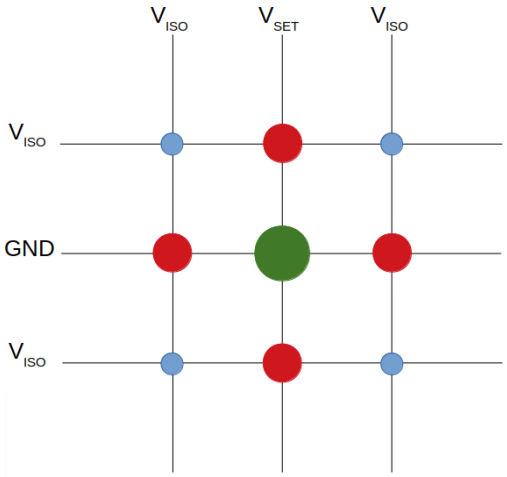
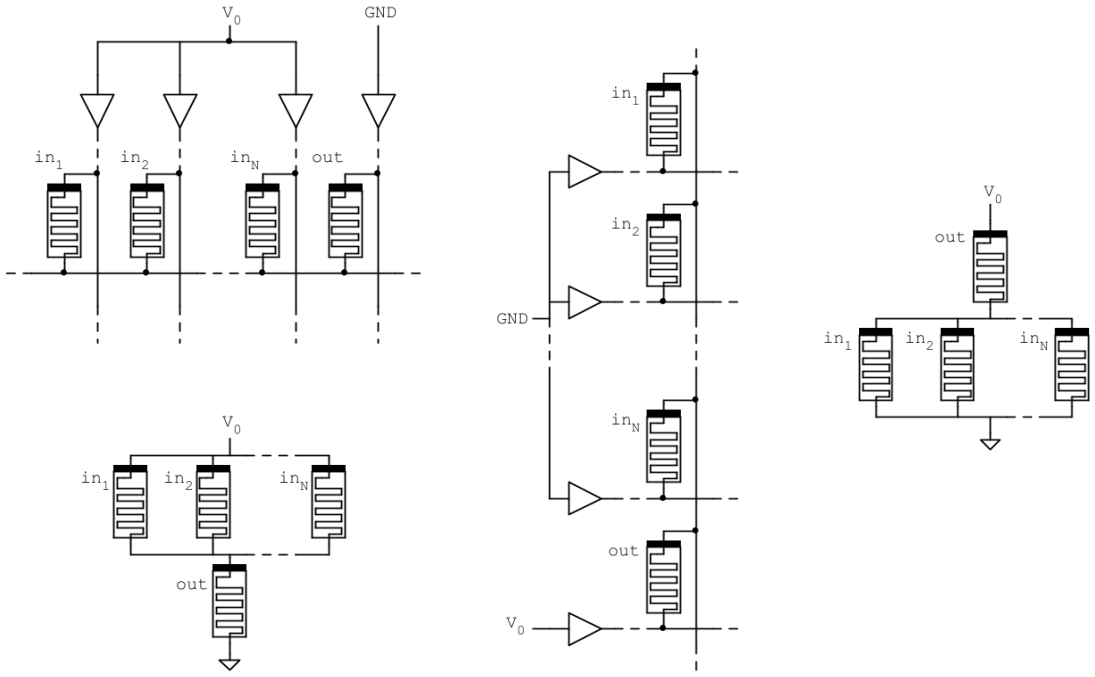


Figure 2.21: Writing operation in the MAGIC crossbar. Green dot: fully-selected cell; red dot: half-selected cell; blue dot: unselected cell[14].

V_O assumes a value according to the states of the input memristors. That is, being V_{OFF}, V_{ON} the memristor thresholds and R_{ON}, R_{OFF} the resistance states, the voltage drop on the output memristor, according to the VTEAM,



(a) MAGIC NOR performed along a row.

(b) MAGIC NOR performed along a column.

Figure 2.22: MAGIC NOR operation in a crossbar [14].

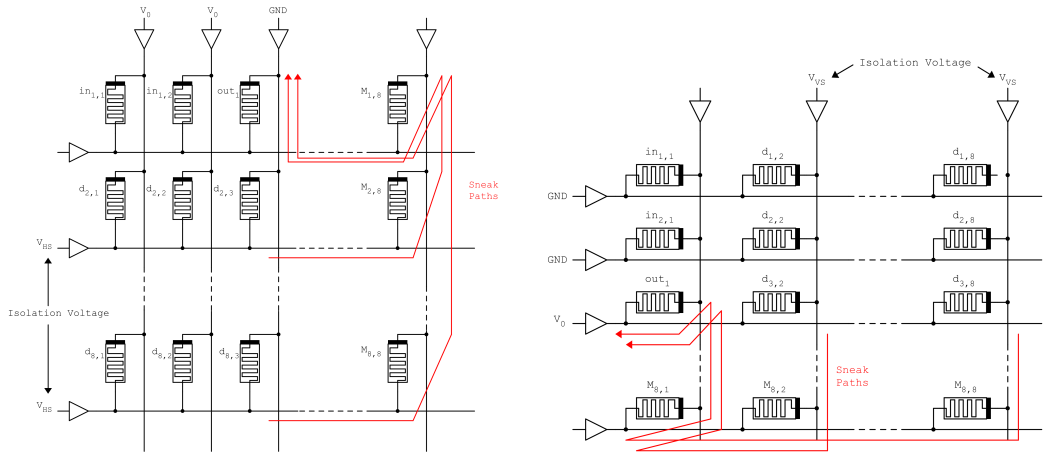
– has to be lower than V_{OFF} if all the inputs are '0';
– has to be larger than V_{OFF} if at least one input is '1'.

Additionally, in order to not switch the input cells, the voltage drop on them has to be lower than V_{ON} when all of them are '0'.
As a consequence, V_O ends up with boundaries:

$$\frac{V_{OFF}}{R_{ON}}\left(R_{ON}+\left(\frac{R_{OFF}}{N-1}\right)||R_{ON}\right)<V_O,$$
$$V_O<\min\left[V_{OFF}\left(1+\frac{R_{OFF}}{NR_{ON}}\right),|V_{ON}|\left(1+\frac{NR_{ON}}{R_{OFF}}\right)\right] \tag{2.12}$$

Finally, regarding Equation 2.12, if N tends to a very large number, the range of V_O is shrunk around V_{OFF}.
Furthermore, the NOR operation with $N=1$ becomes a NOT.

Although MAGIC is a possible approach for Logic in Memory, the issue of sneaky currents has to be dealt with.
In fact, when the MAGIC operation is performed, it is supposed to involve



(a) Sneaky currents in row MAGIC NOR.

(b) Sneaky currents in column MAGIC NOR.

Figure 2.23: MAGIC NOR operation in a crossbar [14].

only a set of cells. That is, isolation voltages prevent other cells to be uncorrectly written.
Nevertheless, as is shown in Figure 2.23, sneaky currents which flow through the output memristor are produced by these voltages, thus the resistance state of the output may be altered so far.
Among the solutions proposed in [16], one of them consist of providing each cell of a transistor to cut off the leakage currents. (1T 1M cell).

Programmable spintronics devices

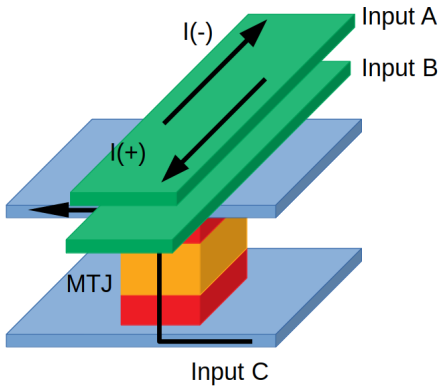


Figure 2.24: 3D model of a programmable spintronics cell [17].

As previously discussed in subsection 2.2.1, the two polarization states of a MTJ are associated to logic values through its resistance state. Therefore, a programmable spintronic device is realized by a MTJ and three electrodes:

- The two input electrodes, here A and B, are placed above the MTJ. These are traversed by polarized bidirectional currents (I_A, I_B) whose magnetic field act to switch the polarization of the MTJ free layer, following the same principle of the spin orbit torque writing in a SOT–MRAM. Only if the currents directions agree, an actual magnetization is forced to the free layer;

- the third electrode, C in Figure 2.24, is auxiliary and attached to the pinned layer. Its purpose is to heat up the aforementioned layer by means of I_C making possible to force an arbitrary magnetization to it. Thus, this establishes the *programmability* of the cell.

In order to fix a convention, here a positive current, hence '1', leads the magnetization to the right; otherwise, a negative current, hence '0', leads the magnetization to the left.
The model is shown in Figure 2.24.

At this point, to show how such a device is able to perform any logic

Set step			
A	B	polarization	Out
1	1		0
1	0		1
0	1		1
0	0		1

Table 2.7: NAND operation with programmable spintronics device.

operation, the most representative Boolean function is analyzed. That is, a spintronics–based NAND.
The NAND function is reached in two steps:

- **Set step**: here both input currents push the magnetization to the left, thus $I_A = I_B =$ '0'; also I_C is on and, as a result, both the free layer and the pinned layer are polarized to the left (parallel polarization, low resistance);
- **Computation step**: the directions of I_A and I_B depend on the values of the inputs; electrode C is unused, so $I_C = 0\,\text{A}$. Then, the only case in which the free layer switches to the right, thus leading to an anti-parallel polarization, is when $I_A = I_B =$ '1'.
  Overall, this circuit fulfills the NAND requirements.

Therefore, the two steps are summed up in table 2.7.

As far as the integration in memory as smart cell is regarded, the presented spintronics–based logic gate can be used as a LiM cell thanks to its storage capability. In particular, a spintronics-based LiM cell is composed by:

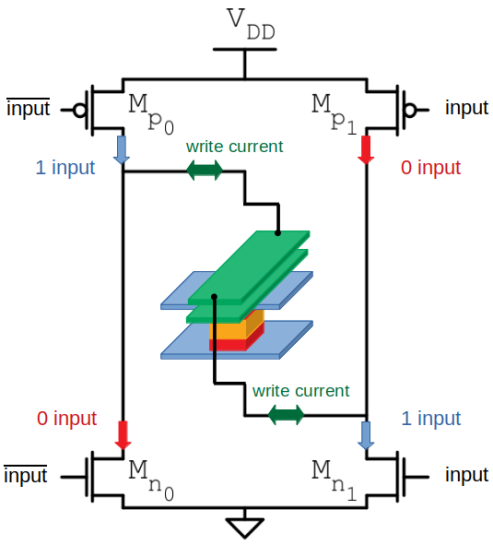
- **Write circuit**: a bidirectional current source is responsible for the magnetization of the SOT–MTJ;
- **Read circuit**: a sense amplifier detects the state of the SOT-MTJ by comparison with a reference current;
- **Logic element**: the cell itsef is used as logic. Moreover, the logic operation is destructive. That is, the inputs must derive from other cells in the crossbar array or from external signals.

Finally, in Figure 2.25 the bidirectional current source and the sensing circuit are respectively described.

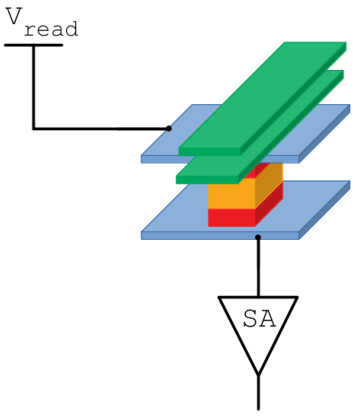
Ferroelectric-capacitor smart cell

Ferroelectric capacitors are exploited as non-volatile elements in what is called *Complementary Ferroelectric-based logic Cell* (CFC). The following description is derived from [12].
A CFC is shown in Figure 2.26. This cell is designed for both storage and logic, according to which phase is executed. Its main parts are:

- two ferroelectric capacitors, which store the data Y in a complementary way;
- the cell pass transistor, which is responsible for the computational feature of the cell;



(a) Bidirectional current source. Here, four transistor in CMOS digital logic switches the direction of the current provided by the supply; then it is driven to the MTJ cell.



(b) Sensing circuit. Here, a sense amplifier compares the current from the MTJ cell with a reference and outputs a digital voltage.

Figure 2.25: Writing and reading circuits for the programmable spintronics-based smart cell [17].

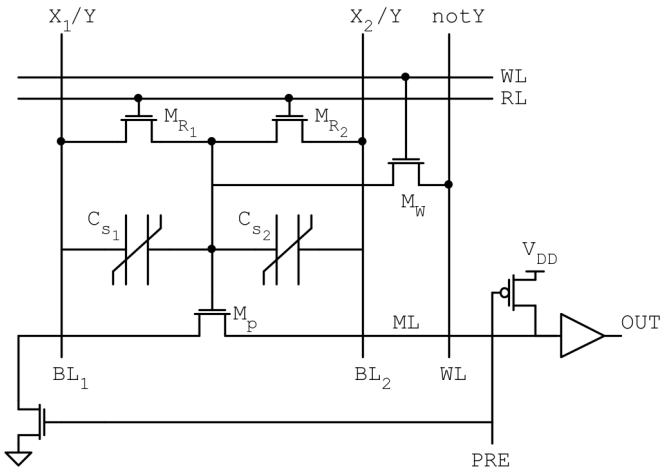


Figure 2.26: Schematic of a complementary ferroelectric-based logic cell [18].

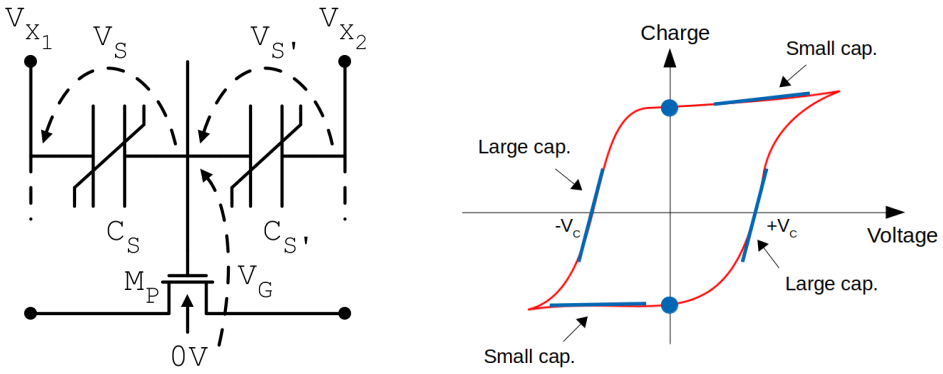
- two bit lines, connected to the external node of each Fe–cap, therefore driven by either the inputs $X_{1,2}$ during a logic phase or the data Y to be loaded during a storage phase;

- a write bit line and a write transistor, allowing to drive the common node of the Fe–caps with $\overline{Y}$;
- two reset transistors, to initialize the common node bringing the Fe–caps to zero voltage drop, a preliminary operation before the logic phase.

Furthermore, the pass transistor of the cell connects it to an external output circuit. That is, an output match line provided with pull–up and pull–down transistors (precharge transistors) which is activated during a logic phase. Indeed, it is part of the cell.

Regarding the different phases that define the functionalities of this cell:

- **Read phase**: the charge induced on the metallic electrodes of the MIM capacitance is sensed by integrating the current over a sweeping voltage;
- **Storage phase**: to load a logic data to the Fe–caps, the two terminals of each of them become controllable by means of the activation of the write transistor.
  Thus, applying $+V_{SR}$ or $-V_{SR}$ which are greater than the critical voltages (see Figure 2.9), the remnant–polarization of the Fe–caps is switched accordingly;
- **Reset phase**: as discussed before, during a reset the Fe–caps are initialized to zero voltage drop, keeping their non-volatile polarization;



(a) Detail of the CFC [18]. (b) Fe-cap hysteresis diagram [18].

Figure 2.27: Detail of the CFC and hysteresis diagram for logic purpose [18].

- **Logic phase**: as far as the logic phase is regarded, a detail of the CFC cell is shown in Figure 2.27, which is sufficient to describe the behavior

of the device in this phase.
After the reset phase, the precharge transistors are turned on and the bit lines are driven by two inputs V_{X_1} and V_{X_2}. Then the voltage drop V_G on the common node of the Fe–caps is dependent on X and Y. In particular:

- if $V_{X_1} = V_{X_2}$, then $V_G = V_{X_1} = V_{X_2}$, regardless of Y;
- if $V_{X_1} \neq V_{X_2}$, Y determines V_G. Being $V_{SS} \equiv$ '0' and $V_{DD} \equiv$ '1', assume that $V_{X_1} = V_{DD}$ (V_{SS}) and $V_{X_2} = V_{SS}$ (V_{DD}). Then, a positive (negative) voltage is transfered to both Fe–caps. That is, recalling the hysteresis diagram in Figure 2.27, the Fe–cap with positive polarization is in small (large) capacitance state while the other one with negative polarization is in large (small) capacitance state.
  Therefore, according to the capacitive divider, the voltage drop is greater on the small capacitance. V_G then turns either on or off the pass transistor.

The results are summed up in Table 2.8. Finally, by covering Table 2.8

Y \ $X_1\ X_2$	'00'	'01'	'11'	'10'
'0'	OFF('0')	ON('1')	ON('1')	ON('1')
'1'	OFF('0')	OFF('0')	ON('1')	OFF('0')

Table 2.8: Switching state of the pass transistor in a complementary ferroelectric-based logic cell [18].

with Karnaugh's maps, it follows:

$$F(X_1, X_2, Y) = X_1X_2 + X_1\overline{Y} + X_2\overline{Y} \tag{2.13}$$

In conclusion, as in Figure 2.28, each individual smart cell is inserted in a larger architecture by means of its pass transistor, in what resembles a wired AND/OR circuit.

Reconfigurable 1FeFET (N)AND-(N)OR smart cell

A smart reconfigurable cell with bitwise (N)AND/(N)OR operations based on a single FeFET is explained in [22]. In fact, the FeFET hysteresis characteristic can be shifted along the voltage axis. In particular, the two main

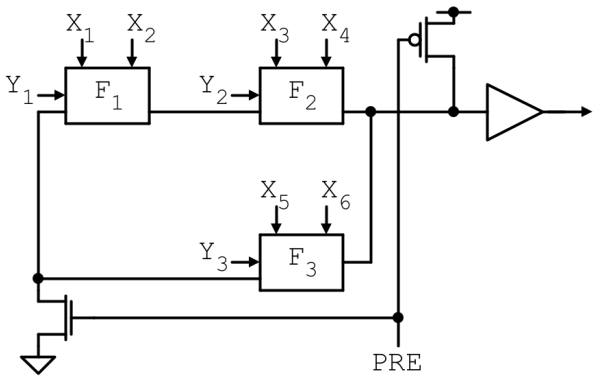


Figure 2.28: Wired AND/OR architecture with complementary ferroelectric-based logic cells [18].

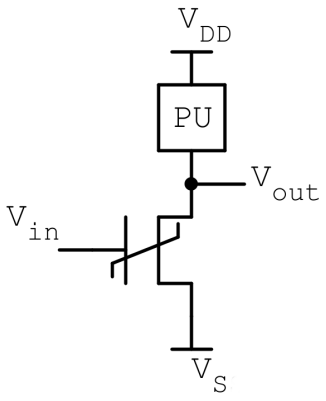
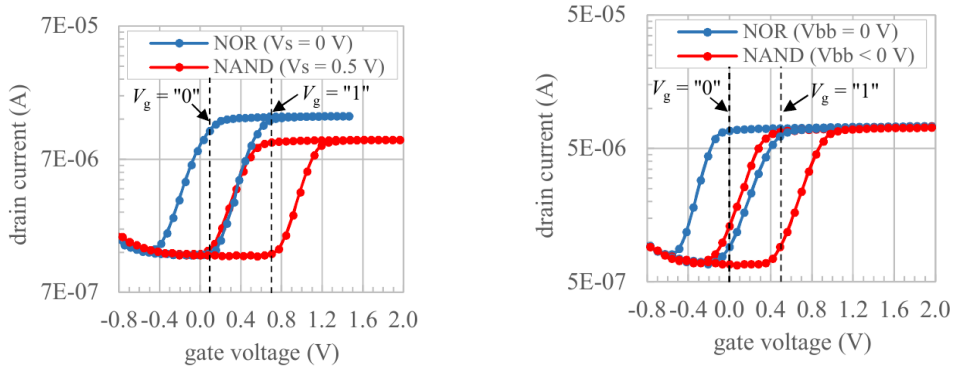


Figure 2.29: Structure of the reconfigurable FeFET NAND/NOR cell [22].

technologies for this applications, the 28 nm HKMG and the 22 nm FD–SOI, show this behavior (Figure 2.30).

The structure of this smart cell is shown in Figure 2.29. It consists of a n–type FeFET in series with a pull–up device. The latter converts the output drain current to a voltage output, while the gate voltage and the polarization state of the gate are the two inputs of the function.
Consequently, as a smart cell, the operations allowed by the circuit in Figure 2.29 are:

- **Write**: a gate voltage higher than the voltage levels required during any logic operation is sufficient to either programming or erasing the cell, in other words to switch from the high threshold to the low threshold and viceversa;

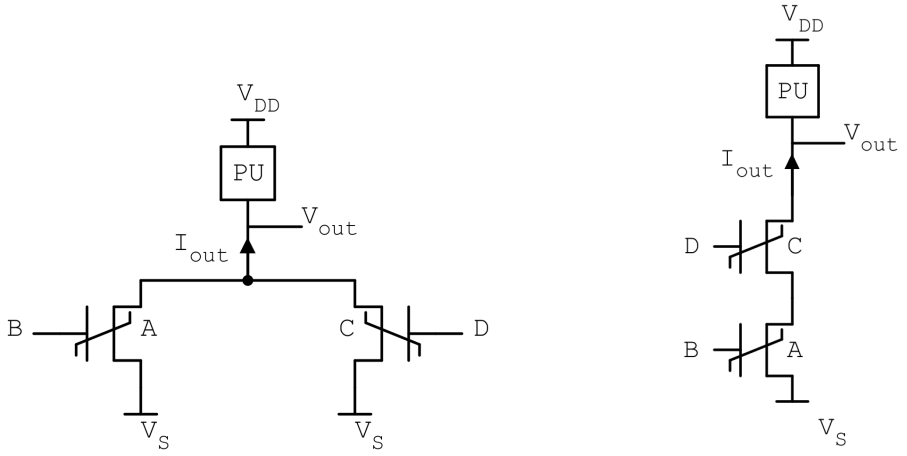


(a) 28 nm HKMG (b) 22 nm FD-SOI

Figure 2.30: Measured I_d-V_g characteristic of the FeFET based NAND/NOR smart cell [22].

- **Read**: referring to Figure 2.30, the memory window is the span between the two thresholds. Therefore, the read operation is accomplished by the application of a low read voltage which, according to the technology, differs depending on which logic operation is configured at the time. The output current is then sensed to read the cell content;
- **Logic**: as aforementioned, this cell can perform a logic function (N)AND-(N)OR between its polarization state and its gate input, where the former has to be non–volatilely stored in a previous step. Then, the high or low threshold corresponds to a logic '0' or '1', respectively.
  Furthermore, once the logic function has been configured, the voltage level of the gate input has to be chosen accordingly to the simulated or experimentally measured technology. By analysing Figure 2.30, it is clear that $V_L = 0\,\text{V}$ and $V_H = 1\,\text{V}$ is a suitable design choice. Thus:
  - **(N)OR**: with the FeFET in NOR confguration, only the combination of low gate input and high threshold leads to cut-off current I_d (OR). Then, low I_d is converted to high drain voltage (NOR). In all the other cases, I_d is high;
  - **(N)AND**: with the FeFET in NAND configuration, only the combination of high gate input and low threshold leads to high current I_d (AND). Then, high I_d is converted to low drain voltage (NAND). In all the other cases, I_d is low.
- **Function reconfiguration**: the I_d–V_g curve of the FeFET can be

shifted along the gate voltage axis, by applying an appropriate source voltage V_s or back bias voltage V_{bb}, depending on the technology.
As depicted in Figure 2.30, if the source and the bulk are grounded the device is in NOR configuration. Otherwise, in the HKMG technology a $V_s = 0.5\,\mathrm{V}$ turns the devices into the NAND operation, while in the FD–SOI technology $V_{bb} < 0\,\mathrm{V}$ is necessary to have a NAND operation.



(a) Two logic–enabled FeFETs connected in parallel, equivalent to a NOR.

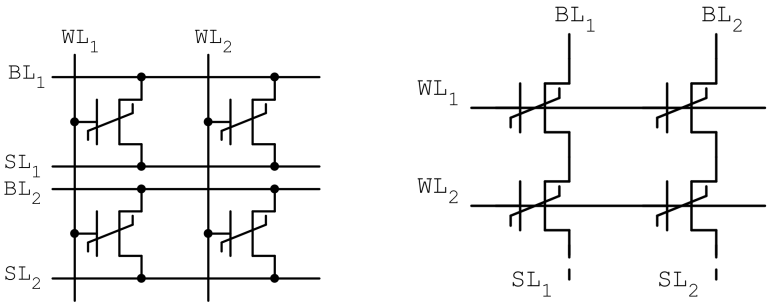
(b) A series connection of two logic–enabled FeFETs, equivalent to a NAND.

Figure 2.31: FeFETs arranged equivalently to a memory array [23].

For what concern the integration of the smart cell in a memory array, the idea is shown in Figure 2.31: here, single logic–enabled FeFETs are combined within parallel and series stacks. That is, the final result is given by summation of drain currents according to the well known Kirchhoff's laws.

Therefore, examples of memory array are shown in Figure 2.32, where two different topologies are proposed, described in [23]. A logic function is obtained between cells in the same row (left) or column (right), where these cells are logically configured by manipulating the source line or the bulk voltage.
Moreover, to realize a XOR function, the array in Figure 2.32 (a) is taken into consideration. Supposing to use one row and two cells, namely cell 1 and cell 2, the steps are the following:

1. The FeFETS are configured in AND mode. This means that either a bias



(a) (N)OR memory array. (b) (N)AND memory array.

Figure 2.32: Integration of one FeFET smart cells in a memory array through bit lines, word lines and source lines [23].

voltage is applied or negative gate voltages are used as inputs (reference to Figure 2.30);

2. If A and B are the inputs of the XOR, $\overline{A}$ and $\overline{B}$ are written to, respectively, cell 2 and cell 1;
3. A and B are driven to, respectively, WL1 and WL2;
4. The output current flowing from BL1 to SL1, representing $A \cdot \overline{B} + \overline{A} \cdot B$, is sensed at the peripherals.

1T-1FeFET smart cell

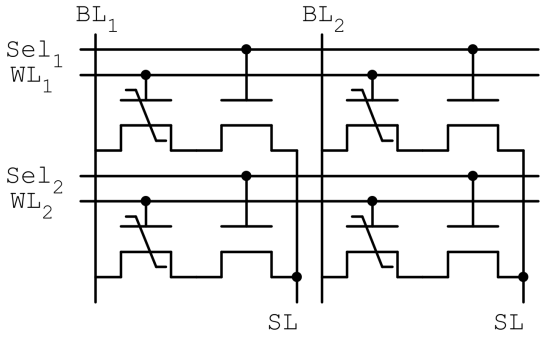
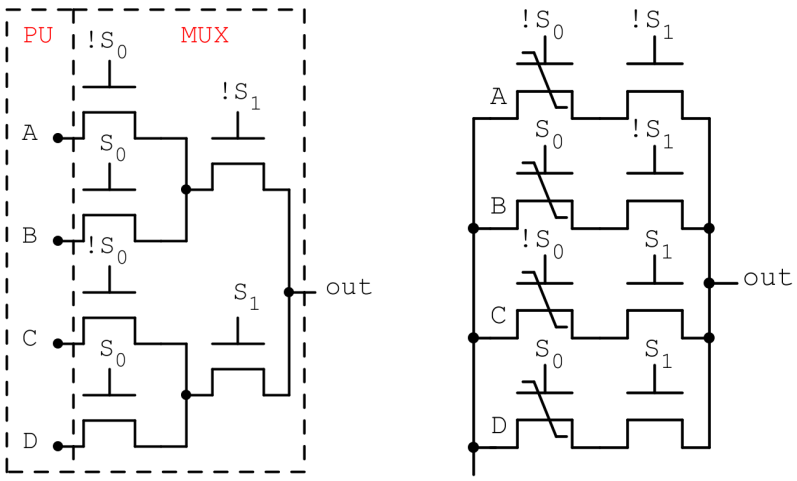


Figure 2.33: Structure of a 1T 1FeFET smart cell [26].

In this version of FeFET smart cell, the single FeFET storing the information bit is accompanied by a series MOSFET transistor, named selector transistor, and introduced in [26].

The integration in the memory array is the same as the previous case, including all the available operations such as writing, reading, logic. An example of it can be viewed in Figure 2.33. This structure is also known as 2T–NOR memory array since the cells are connected in parallel. Moreover, the series selector transistor can include a logic AND inside the cell.

The first example of LiM achievable with this array is a 2–inputs Look Up



(a) Standard LUT.

(b) LUTMUX integrated into the array.

Figure 2.34: Integration of a LUT in both standard transistors and in a 1T 1FeFET smart array [26].

Table (LUT). As shown in Figure 2.34, the standard 2-input LUT is composed by a programmable unit and an output multiplexer. It is the most straightforward implementation of a 2–inputs LUT, where the MUX select signals are the inputs of the boolean function.

In spite of that, the 1T–1FeFET smart array merges those two parts by means of four cells connected to the same bit line and source line. The boolean inputs and their negative, in all the possible combinations, are sent to the word lines and the selection lines. That is, the boolean function is reconfigurable by arbitrarly programming or erasing the content of the FeFETs.

Moreover, the second example is a Full Adder. Precisely, the Full Adder implemented in the smart array in Figure 2.35 takes two rows for a total of ten cells allocated. The concept behind is simple, for which the inputs are distributed among all the cells content and the word lines, and the two output (sum and carry out) are sensed from the two bit lines. In particular,

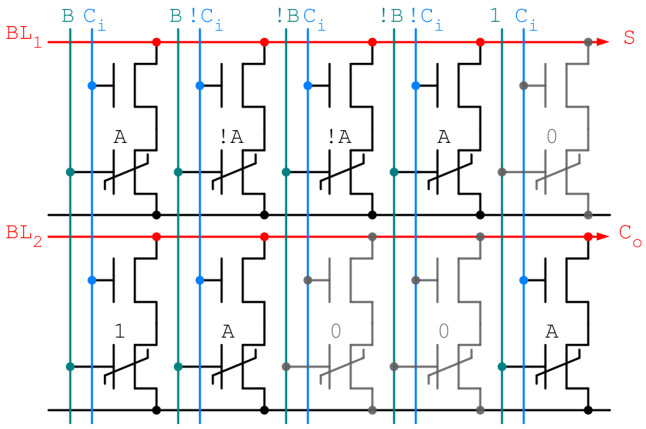


Figure 2.35: Full Adder realized in a 1T 1FeFET smart array. [26]

the logic functions performed are:

$$S = B \cdot (C_i \cdot A + \overline{C_i} \cdot \overline{A}) + \overline{B} \cdot (C_i \cdot \overline{A} + \overline{C_i} \cdot A) \tag{2.14}$$

$$C_o = B \cdot C_i + A \cdot B \cdot \overline{C_i} + A \cdot C_i \tag{2.15}$$

Thus it is demonstrated, as aforementioned, that the selector transistors is exploited to insert an AND in the circuit.

Complementary FeFET TCAM

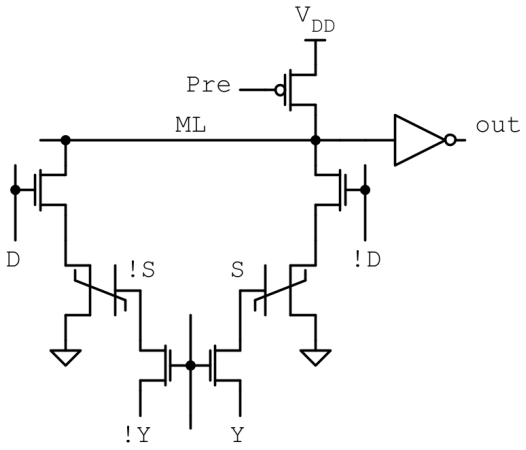


Figure 2.36: FeFET-based core cell for NOR-type TCAM [27].

The wide versatility of FeFETs as memory cells introduces this device also in the field of the Content Addressable Memories (CAM). The implementation of a ternary CAM (TCAM) cell is shown in Figure 2.36 and refers to the work in [27].
A TCAM is conceived to compare a word given as input to the words stored in the memory to find a match. Therefore, the store operation is conducted by means of a tri–state word line connected to the FeFETs gates. The attribute ternary refers to the adding of a third state *"don't care"* to the hit and miss states. In particular, the comparison is based on the following assumptions:

- Two FeFETs are connected in parallel, storing the bit and its complementary;
- the FeFETs are connected to a match line by means of two pass transistors which are controlled by the input data. That is, it is a NOR-type match–mismatch detection;
- the match line is precharged to V_{DD} before the detection and it is discharged to ground in case of miss, otherwise it keeps V_{DD}.

Dynamic current mode FeyFET smart cell

The Dynamic Current Mode Logic (DyCML) style, described in [27], aims at reducing dynamic power by limiting the output swing, at the cost of doubling the number of necessary transistors.
In Figure 2.37 a DyCML is implemented to build a non–volatile LiM, which consist of:

- A clock–controlled pull-up network (top center) responsible for pre–charging and latching to mantain the output even after the logic operation;
- a logic network to obtain the desired arbitrary function;
- a non-volatile cell based on two complementary FeFETs;
- a dynamic, clocked current source.

This FeFET–DyCML is based on the use of a differential pull–down logic network which, combined with the FeFETs cell, generates the corresponding complementary outputs. Furthermore, while the inputs are respectively the content of the cell and one external, the flexible design of the wiring connections of the logic network enables various logic functions, such as NOR,

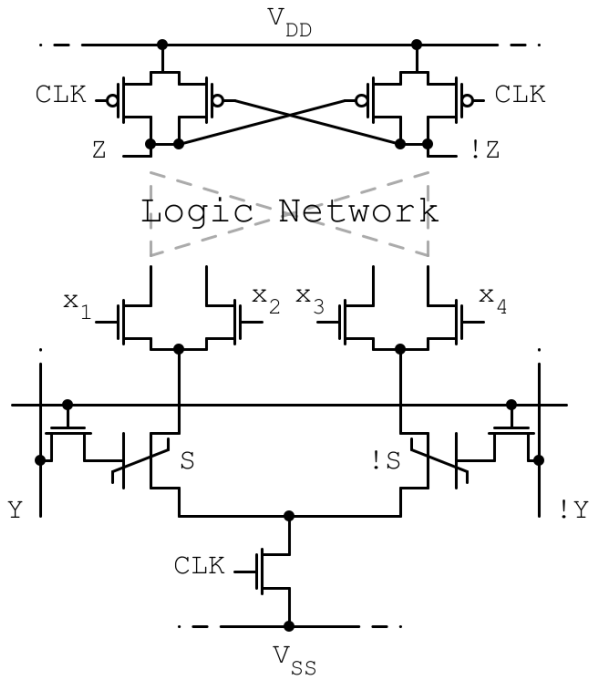
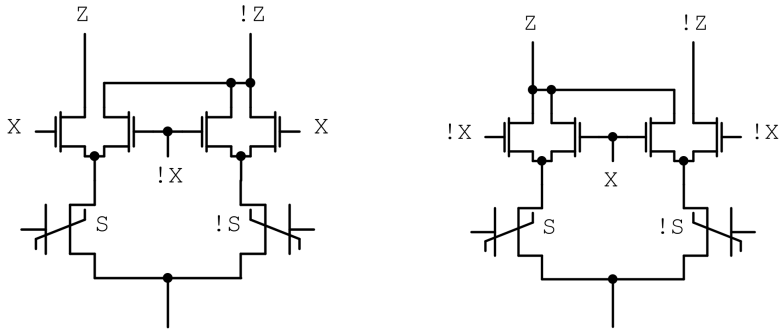


Figure 2.37: General circuit structure of a FeFET-based DyCML LiM cell [27].

NAND, and more complex ones as the Full Adder.
In addition, the writing operation needs a tri–state word line to write the FeFETs content, as in the previous FeFET TCAM.
The first two examples of a DyCML used in LiM mode are the NAND and



(a) NAND logic network of a DyCML.

(b) NOR logic network of a DyCML.

Figure 2.38: Examples of FeFET-DyCML logic functions [27].

NOR functions, as shown in Figure 2.38. Here, the two FeFETs excludes each

other and only one branch at time is activated. Then, the shape of the logic network is composed by couples of semi-parallel transistors, controlled by the external input, and the wired inteconnections estabilish the logic function. Moreover, a more complex example is depicted in Figure 2.39, where a Full

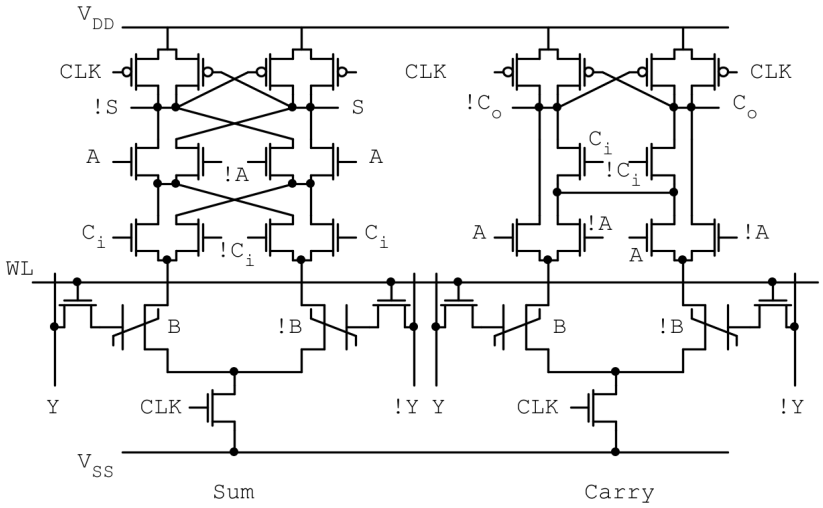


Figure 2.39: Schematic of a FeFET-based DyCML LiM 1-bit Full Adder [27].

Adder is realized by means of 4 FeFETs, 28 MOSFETs (24 for DyCML, 4 for allowing the writing operation).
In this architecture the output are achieved by stacking several stages of transistors couples, with adequate wired interconnections.
In spite of the previous examples, here the inputs are dislocated among the FeFETs and the transistors of the logic network, and two smart cells are required to compute the sum and the carry out. That is, the left cell in Figure 2.39 is configured to perform a triple XOR while the right cell is configured to perform the carry out function.

Dynamic Logic FeFET smart cell

FeFETs–based Dynamic Logic (DL) LiM is based on the high I_{on}/I_{off} ratio of the FeFETs, allowing them to be trustly modelized with either open circuits or short circuits. Moreover, Dynamic logic gates and cells becoms useful when low area circuits are preferred.
A DL smart cell, in Figure 2.40, derives from the DyCML smart cell by employing only one branch of logic network, supported by a single FeFET to store a bit. In particular it is composed by:

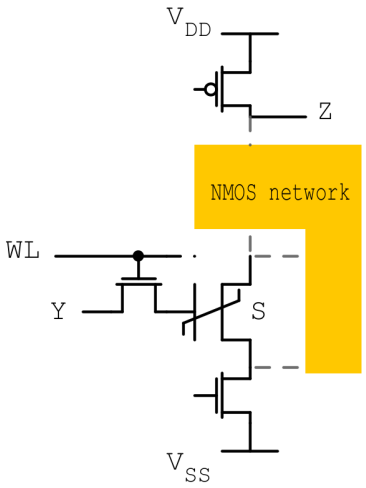


Figure 2.40: General structure of FeFET-based dynamic logic circuits[27].

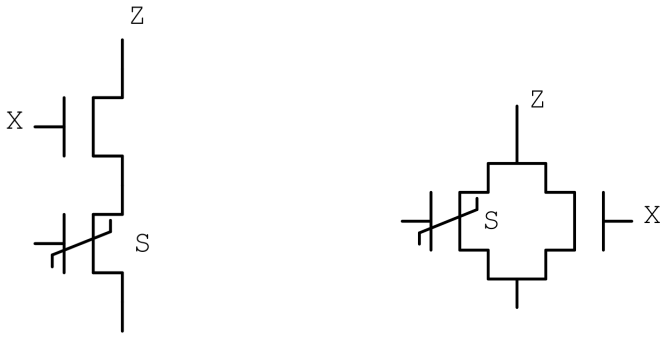
- a pull–up network made by a simple clocked p-type MOS transistor;
- a n–type MOS pull-down logic network that executes the logic function;
- a clocked n–type MOS transistor to discharge the network to ground;
- a FeFET–based cell.

Furthermore, as in the previous FeFET–based circuits, the writing operation is conducted by a tri–state line. The topology of the logic network is different for every logic function, as can be seen in the following examples.

Therefore, the NAND and NOR functions are the examples with the smallest number of transistors. (Figure 2.41). Their structure is composed by a series and a parallel between a FeFET and a MOSFET, respectively. In particular, the NAND configuration resembles the 1T–1FeFET smart cell. Indeed, the working principle is the same as in section 2.2.2.

Nevertheless, a more complex logic configuration is depicted in Figure 2.42. Here, to achieve the two desired outputs (sum and carry out), more than one stage are stacked on the top of each other. Besides, three smart cell are used, all of them containing the same input bit.

Once the outputs are set by the logic pull–down network, the presence of several stages generates voltage drops in the final output dynamic. That is, an output inverting stage is added, at the expense of two additional MOSFETs per output signal.



(a) NAND logic network of a dynamic logic smart cell.

(b) NOR logic network of a dynamic logic smart cell.

Figure 2.41: Examples of FeFET-DL logic functions [27].

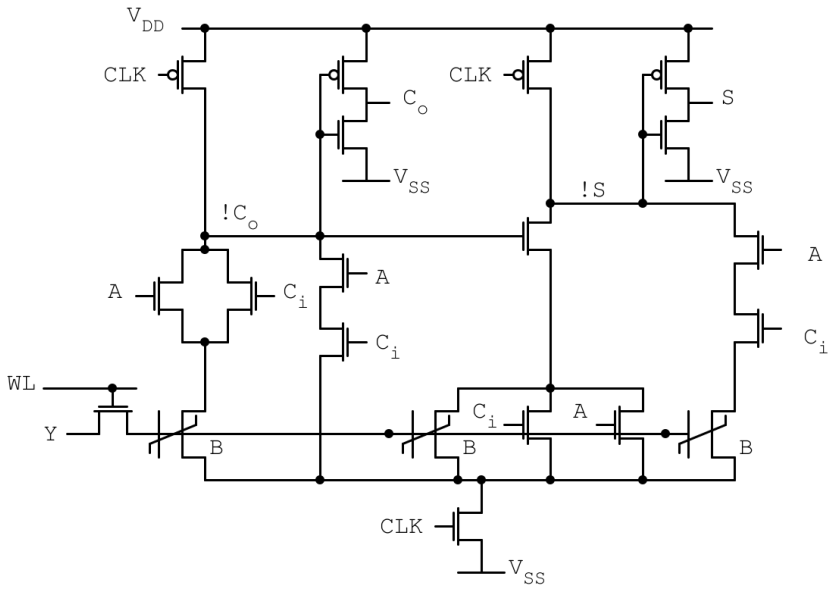


Figure 2.42: Schematic of a FeFET–based dynamic logic 1 bit Full Adder [27].

Chapter 3

FeFET SPICE model

Overview

The purpose of this chapter is to describe and characterize a SPICE FeFET model, which is intended to be used to characterize, first, a conventional memory array and, second, several LiM cells.
As discussed in subsection 2.2.2, there exist several methods to obtain both storage and logic features with one or more FeFETs.
Therefore, the first step is to refer to a physical model for the FeFET, derived from [32], getting the parameters from experimental results in literature, and to translate it to a computational model and finally to a SPICE model. Then, the model is simulated and data are extracted, shown and discussed.

3.1 Physics–based FeFET model

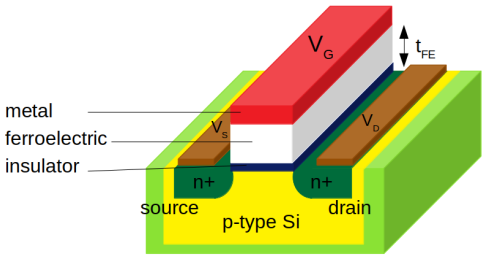


Figure 3.1: 3D model of a n–type FeFET [32].

Different mathematical approaches to the FeFET model exist, as far as the behavior of the internal polarization of the ferroelectric material in the gate stack is concerned. In particular, the gate stack that is examined is shown in Figure 3.1. Here, a Metal Ferroelectric Insulator Semiconductor (MFIS) stack composes the gate of the transistor, in which the total number of carriers in the channel is controlled by the behavior of the gate.
Moreover, any mathematical model characterizes the device by means of parameters which depend on the chosen ferroelectric material, and these are usually extracted experimentally, as described in [32].
Furthermore, the two most used physics–based models are the Landau–Khalatnikov (LK) equation and the Preisach model.

3.1.1 Landau-Khalatnikov equation

The time-dependent LK equation is a single–domain approximation which describes the correlation between the applied electric field and the internal charge density of the gate. From [33], this equation is non-linear:

$$E - \rho\frac{dP}{dt} = \alpha P + \beta P^3 + \gamma P^5 \tag{3.1}$$

where α, β and γ are the static parameters of the ferroelectric and ρ is the kinetic coefficient.
Then, if t_{FE} and A_{FE} are, respectively, the thickness and the area of the ferroelectric, let $V_{FE} = t_{FE} \cdot E$ be the voltage across the ferroelectric and $Q = A_{FE} \cdot P$ be the total charge. That is, Equation 3.1 can be rewritten as:

$$V_{FE} = R_{FE}I_{FE} + C'_{FE}Q + C''_{FE}Q^3 + C'''_{FE}Q^5 \tag{3.2}$$

where $R_{FE} = \rho\frac{t_{FE}}{A_{FE}}$, $C'_{FE} = \alpha\frac{t_{FE}}{A_{FE}}$, $C''_{FE} = \beta\frac{t_{FE}}{A^3_{FE}}$ and $C'''_{FE} = \gamma\frac{t_{FE}}{A^5_{FE}}$. The form of the derived Equation 3.2 recalls a circuit scheme composed by a resistor and a non–linear capacitor. The complete circuital scheme is depicted in Figure 3.2, which features also a MOSFET and a conventional capacitance in parallel to the LK–gate, $C_0 = \epsilon_0\frac{A_{FE}}{t_{FE}}$.
That is, in the SPICE model it is assumed that the LK–based gate is in series to the MOSFET gate.
Nevertheless, in the SPICE model, a subcircuit is needed to generate the non–linear voltage–charge relationship, which is not recalled in Figure 3.2. Therefore, a Verilog–A model is derived and is compatible to the majority of the SPICE simulators, such as Cadence Virtuoso.

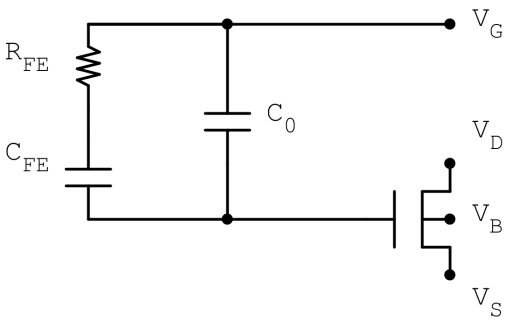


Figure 3.2: proposed circuit/SPICE model for FeFET [32].

Verilog–A script

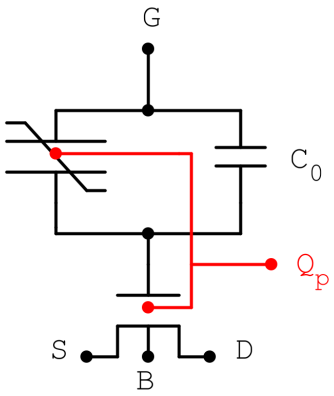


Figure 3.3: FeFET circuit scheme equivalent to the Verilog-A script.

The scheme in Figure 3.3 shows two distinct parts —the gate part and the MOSFET part— which communicate through a dummy node. That is, the latter carries the information about how much charge density is present in the ferroelectric gate.
The introduction of the dummy node, which is considered a fake voltage node in the simulator, is sufficient to fix the same amount of charge between the transistor gate and the ferroelectric gate.
As far as the ferroelectric gate is concerned, the following Verilog–A extract describes its behavior:

```
1 V(fecap)   <+alpha*tFE*(V(qg_as_v)*1e-6) + beta*tFE*pow((V(
    qg_as_v)*1e-6),3.0) + gamma*tFE*pow((V(qg_as_v)*1e-6),5.0)
    ;
```

```
2    V(fecap)    <+ rho*tFE*ddt(V(qg_as_v)*1e-6);
3    I(c0) <+ C0*ddt(V(c0));
```

Listing 3.1: Ferroelectric capacitor in Verilog–A description. Reference to Figure 3.3.

In the code above:

- the first line describes the non–linear behavior;
- the second line describes the resistive behavior;
- the third line describes the conventional capacitive phenomenon of a dielectric device.

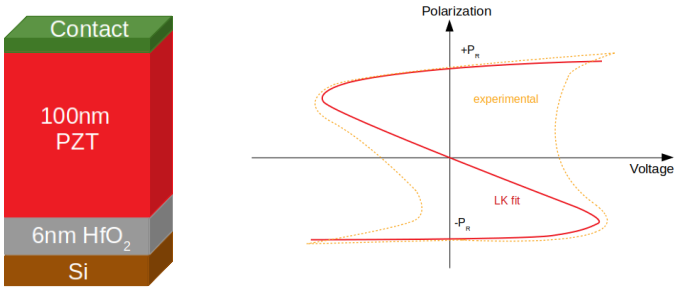
Moreover, `qg_as_v` is the name of the dummy node.
The Verilog–A code for the MOSFET is a virtual–source (VS) based self-consistent transport/capacitance model for silicon MOSFET, developed by the Purdue University. It is reported entirely in the Appendix.
Furthermore, reference to both source codes are found in [29].

3.1.2 Preisach model

The Preisach model is a multi–domain model and its modeling framework, presented in [28], is composed of two subcomponents, similarly to the LK model in subsection 3.1.1. In this case, two equations governing charge conservation and voltage division must be solved simultaneously.
Therefore, the determination of the gate charge density is dependent on the ferroelectric history, switching dynamics and minor loop trajectory.
Contrarily to the LK model, this description translates to a more accurate computational model, although it is difficult to implement in SPICE or Verilog–A language being an algorithmic model and not only a mathematical model, according to [28]. That is, for the following sections the single domain LK model will be employed.

3.2 FeFET experimental calibration

From the studies conducted in [32], to calibrate the model described by the LK equation (Equation 3.1), the parameters α, β, γ and ρ have to be experimentally determined since these act as fitting coefficients to reduce the differences between the ideal model and the real device.



(a) Gate stack experimentally measured.

(b) Experimental P-E loop and LK fit.

Figure 3.4: Calibration of the LK model with experiments [32].

Therefore, in [32], a 100 nm lead zirconium titanate (PZT) film has been grown on hafnium oxide (HfO_2) buffer and silicon substrate using pulsed vapor deposition.
Then, the static LK coefficients have been extracted from the experimental P–E loop in Figure 3.4, while the value of ρ has been calculated from the polarization switching time.
As a result, the following parameters have been set in the Verilog–A model:

$$\alpha = -1.05 \times 10^{9}\,\mathrm{m\,F^{-1}} \tag{3.3}$$

$$\beta = 1 \times 10^{7}\,\mathrm{m^{5}\,F^{-1}\,C^{-2}} \tag{3.4}$$

$$\gamma = 6 \times 10^{11}\,\mathrm{m^{9}\,F^{-1}\,C^{-4}} \tag{3.5}$$

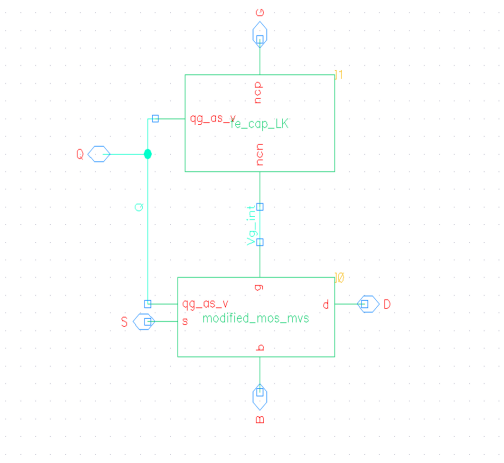
$$\rho = 0.25\,\Omega\,\mathrm{m} \tag{3.6}$$

3.3 FeFET simulation and characterization

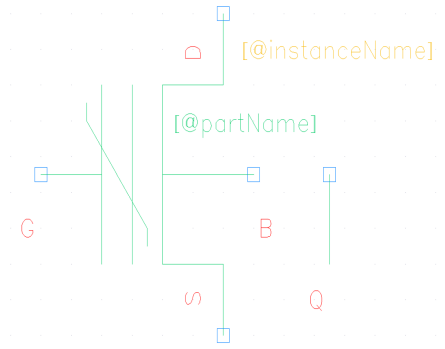
3.3.1 Schematic and testbench

An n–type FeFET schematic has been created in the Cadence Virtuoso CAD environment starting from the Verilog–A description in subsection 3.1.1.
In particular, the two Verilog–A models has been converted into building blocks as shown in Figure 3.5. Also, the symbol is shown.

Therefore, the model has been tested in order to characterize, for any writing operation, its actual behavior in terms of frequency response, I_{ON}/I_{OFF}

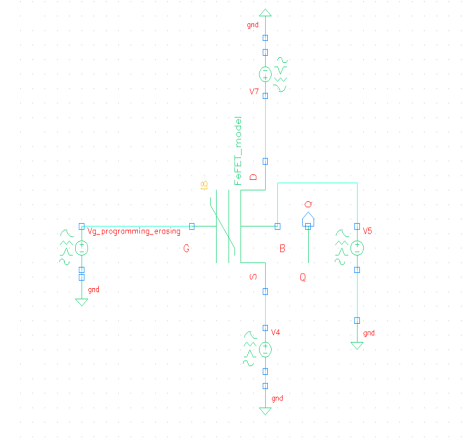


(a) n–type FeFET schematic in Cadence Virtuoso CAD Tool.

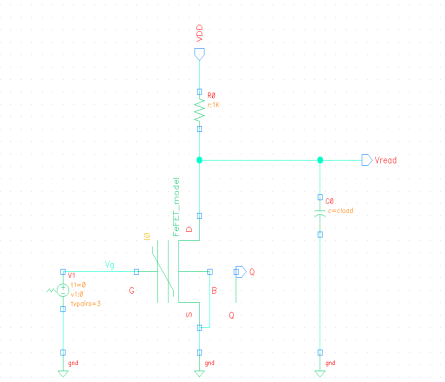


(b) n–type FeFET symbol in Cadence Virtuoso CAD Tool.

Figure 3.5: n–type FeFET in Cadence Virtuoso CAD Tool.



(a) FeFET model testbench for writing operation.



(b) FeFET model testbench for reading operation.

Figure 3.6: FeFET model testbenches.

ratio and performance when technological parameters are varying, such as the thickness of the ferroelectric material.
For this purpose, a testbench has been created in Virtuoso and is presented in Figure 3.6.
Furthermore, the model has also been tested for the reading operation and the testbench for this purpose is depicted in Figure 3.6.

3.3.2 The impact of the ferroelectric thickness

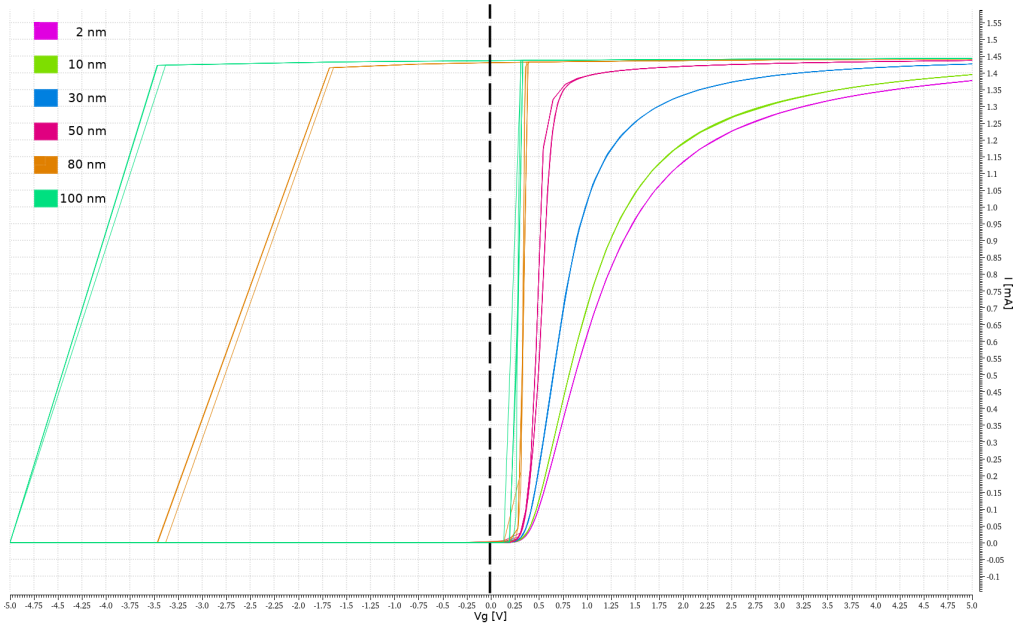


Figure 3.7: Simulation of the hysteresis of the FeFET gate polarization with varying thickness of the ferroelectric layer. I–V characteristic.

An aspect not covered in section 3.2 is how the thickness of the ferroelectric layer (t_{FE}) impacts the model behavior. In section 3.2, a 100 nm thickness is chosen to perform the data extraction.
That is, this value appears to be the suitable choice for the fitting parameters used in the model.
Nevertheless, simulations have been conducted with different values of thickness in order to demonstrate and justify the choice about that value.
The results of this simulation are reported in Figure 3.7, which refers to a pulsed gate voltage with frequency of 10 kHz and peak amplitude of 5 V. V_{DS} is set to 300 mV.
Here, the output characteristic I–V recalls the conventional MOSFET for low t_{FE}, losing the hysteresis feature which is being exploited in this work to use the FeFET as a memory device.
On the contrary, $t_{FE} = 100\,\text{nm}$ leads to hysteresis and will be the current choice for the following sections.
Finally, a dashed lined has been plotted to highlight the behavior of the current conduction of the device at $V_G = 0\,\text{V}$. That is, the model exhibits two distinct thresholds which will be exploited to read the information stored in

the FeFET.

3.3.3 Hysteresis frequency response

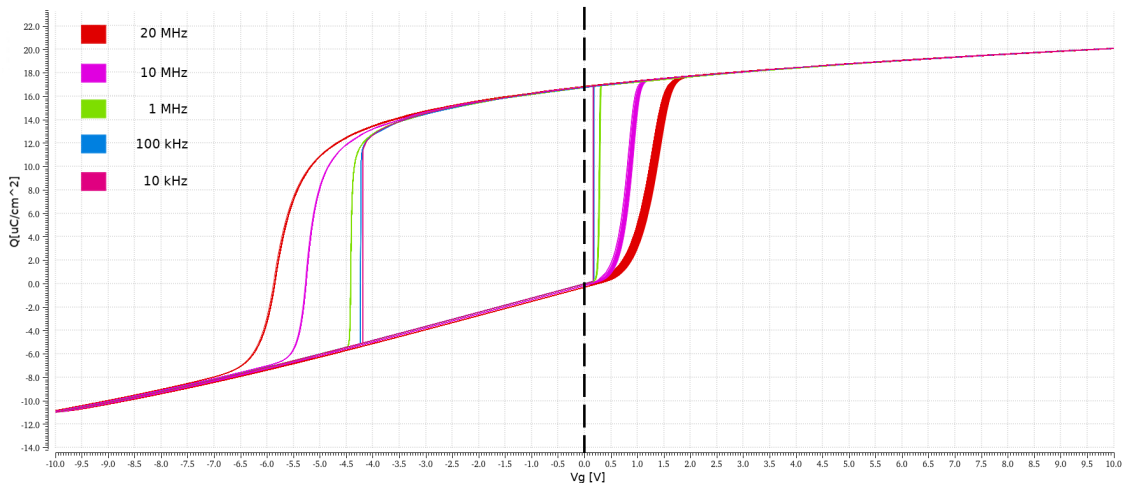
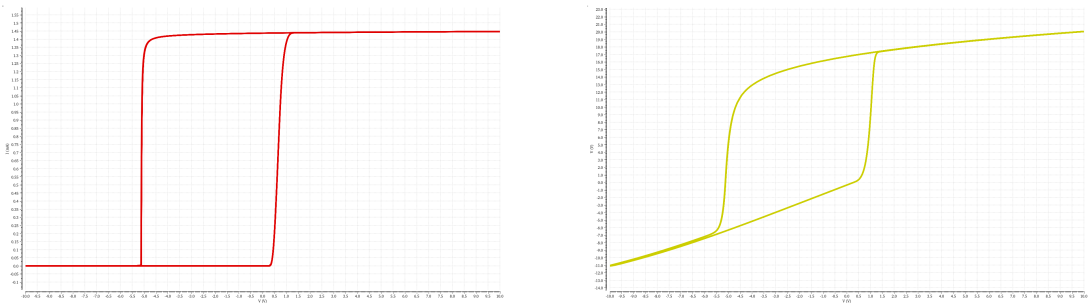


Figure 3.8: Simulation of the hysteresis of the FeFET gate polarization with varying periods of the gate voltage. Polarization vs. gate voltage.

Another test performed with this FeFET model is the frequency response. In other words, when the FeFET is used as building block for the composition of a memory cell or LiM cell, it is useful to know how it reacts to different working frequencies applied to the circuit.
Therefore, Figure 3.8 is the result of the simulation with a set of different frequencies, from 10 kHz to 20 MHz.
Here, the polarization of the gate is plotted versus the gate voltage, which is a triangular wave with peak amplitude of 10 V, and a change in both the high and low threshold is visible.
This behavior is due to the fact that the internal polarization is affected by a delay when exposed to a variation, and consequently faster signals on the gate require higher voltage levels to accomplish the switching of the polarization. That is, both the low and high thresholds become larger, in absolute value, when the frequency rises.



(a) I–V output characteristic at 10 MHz. (b) Q–V output characteristic at 10 MHz.

Figure 3.9: FeFET hysteresis simulation at 10 MHz.

3.3.4 Other measurements

Writing example

In Figure 3.9 it is shown an example of response of the FeFET to the application of a 10 MHz triangular wave to the gate.
Thereby, Figure 3.9(b) exhibits the two distinct equilibrium points of the gate polarization.
Moreover, Figure 3.9(a) is a plot of the drain current where the two thresholds are visible. With $V_{DS} = 300\,\text{mV}$, the on current results in $I_{ON} \simeq 1.5\,\text{mA}$.

Reading example

Referring to Figure 3.6(b), a single–ended reading circuit has been set up to demonstrate an example of reading of the content of a n–type FeFET.
That is, a pull–down resistor R_{OUT} translates the drain current into a voltage drop such that the output voltage is driven to $V_L \simeq 0\,\text{V}$ when the FeFET is in low threshold state.
Moreover, a capacitive load has been considered.
The result of the reading is depicted in Figure 3.10, where the content of the cell is the logic '1' and the output voltage goes to V_L.
Otherwise, reading a logic '0' is not useful due to the fact that the output voltage would remain pulled–up.

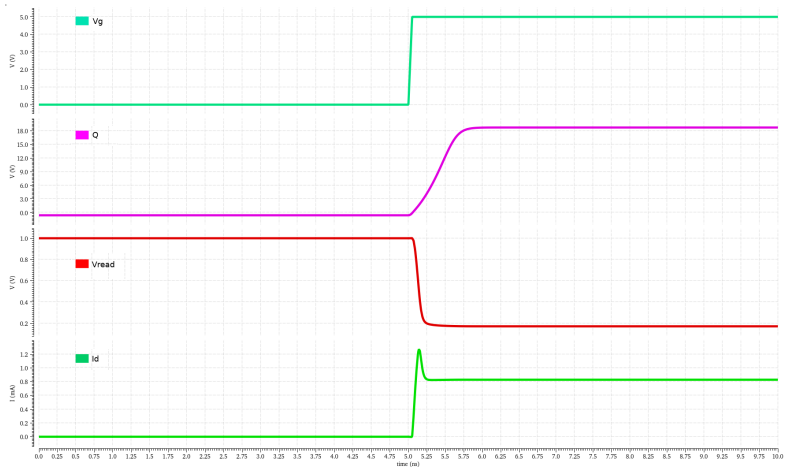


Figure 3.10: Reading waveforms of the FeFET model with $R_{OUT} = 1\,\text{k}\Omega$ as pull–up resistor and $C_{LOAD} = 100\,\text{fF}$ as load capacitance.

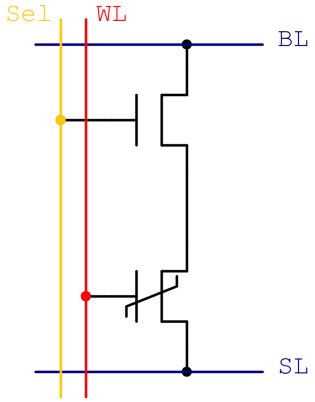


Figure 3.11: FeFET–based memory cell.

3.4 Basic FeFET–based memory cell

The memory cell in Figure 3.11 is composed by an n–type FeFET and a conventional n–type MOSFET. In particular, the FeFET is responsible for data storage, while the MOSFET acts as a selector, plugging and unplugging the cell to the bit line of the array.
As far as logic is concerned, this cell is not capable of LiM. Nevertheless, this cell is the basic block of memory arrays of varying dimensions which are characterized in chapter 4.
Moreover, the functions of this cell are here summed–up:

- **Write**: the cell content can be written through a word line (WL) which is connected to the gate of the FeFET. '0' and '1' are associated to, respectively, high conduction threshold state and low conduction threshold (Figure 3.9, to have a reference). During this operation, the selector is off.
  Moreover, a single WL is shared across a word of the array.

- **Read**: the cell content can be sensed through a bit line (BL) while the selector is on. That is, the select signal is supposed to be shared across a single word of the array.
  For what concerns the WL, as discussed in section 3.3, the reading voltage on the gate of the FeFET is 0 V. Then, the cell can either show low or high conduction.

Chapter 4

FeFET–based memory array

Overview

As aforementioned in section 3.4, a memory architecture is generated starting from the basic cell shown in Figure 3.11.
Moreover, due to the fact that the FeFET recalls the behavior of a Floating Gate MOSFET (FGMOS), two topologies have been investigated at the purpose of creating a memory array. That is, the NAND FLASH topology and the NOR FLASH topology.

4.1 The topology

Referring to [39], the NAND and NOR architectural solution have been inquired if suitable to be applied to the case of the FeFET. Indeed, a FeFET can be either in the state of conduction or cut off as well as a FGMOS does. Moreover, the writing operation occurs by the application of a large gate voltage in both devices.
Finally, both devices present a current output to be sensed. Thus, the FLASH memory design has been the most inspiring for the purposes of this work.

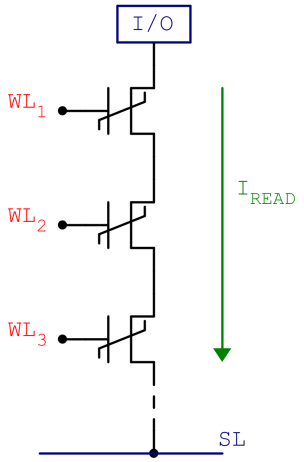


Figure 4.1: A column of the NAND–FeFET array [39].

4.1.1 NAND architecture

Suppose to build a columna of FeFETs as in Figure 4.1, inspired by the NAND FLASH architecture in [39].
Thereby, each FeFET in the column: acts as a cell, is driven by its own word line and shares the same source line (SL) at the bottom of the column.
Therefore, during a reading operation of a cell inside the column, a reading voltage of 0 V is applied to the FeFET to be read. Then, all the other FeFETs in the string are supposed to be forced to conduction with the application of a *coercitive* voltage, to properly obtain the reading.
Nonetheless, this operation is unfeasible, due to the nature of the FeFET gate polarization. That is, there exist no coercitive voltage that leads the FeFET in conduction without programming the FeFET itself.
Thus, reading in a NAND–topology FeFET memory array would be destructive for the data stored.
An example of how the internal polarization would be altered is shown in Figure 4.2. Here, the gate voltage is set to a ramp starting from -5 V to 5 V and lasting 20 ns. Moreover, the sensing circuit is the same as in Figure 3.6. The initial content of the cell is '0', from Figure 4.2, and the output value is correctly '1'. Nevertheless, as V_G increases, the output falls to '0' meaning that the cell is forced to conduction, but also the polarization is altered and the data stored is destroyed.

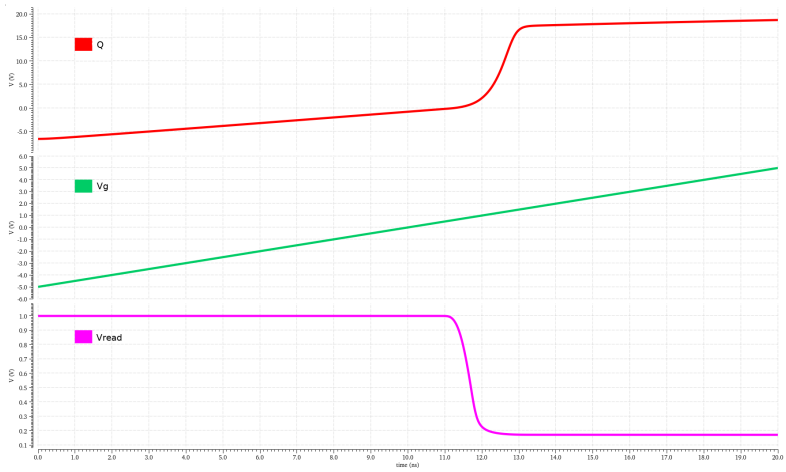


Figure 4.2: Simulation of a reading operation in a NAND–topology FeFET memory column.

4.1.2 NOR architecture

Since the NAND topology has been discarded according to subsection 4.1.1, the NOR topology is adopted for the rest of this work. In particular, the memory cell is the same as shown in Figure 3.11.
Here, each cell in a memory word is connected to its own BL and SL, thereby each cell has its own sensing circuit and one word per time is read.
Therefore, the reading of a FeFET does not affect the data stored in the other cells, as explained in subsection 4.2.4.
Moreover, the NOR topology is inspired by the tractation in [39].

4.2 NOR FeFET memory array: schematics

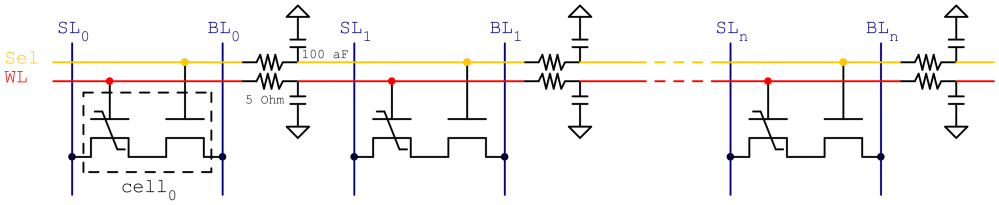


Figure 4.3: A word of the FeFET–based NOR memory array with n cells.

As discussed in subsection 4.1.2, a NOR memory array with the basic FeFET cell is presented. In particular, a word of this memory architecture

is depicted in Figure 4.3.
Thus, bit lines and source lines are dedicated to each single cell in a word, while word lines and select lines are shared across the same word. The words of this proposed memory are organized in columns. Contrarily to that, Figure 4.3 shows the word adapted as a row for the sake of clearness of the figure.
Furthermore, with this topology, a word (column) is either read or written per time.

Interconnections parasitics

The non–idealities of the memory array are modelized in Figure 4.3 by means of parasitic *RC* at the interconnections between cells. In particular, the values for the parasitic resistance and capacitance are, respectively, 5 Ω and 100 aF.
Moreover, these parasitics affect the worst case delay measurements, as it is discussed in subsection 4.5.1.

4.2.1 8x8 memory array

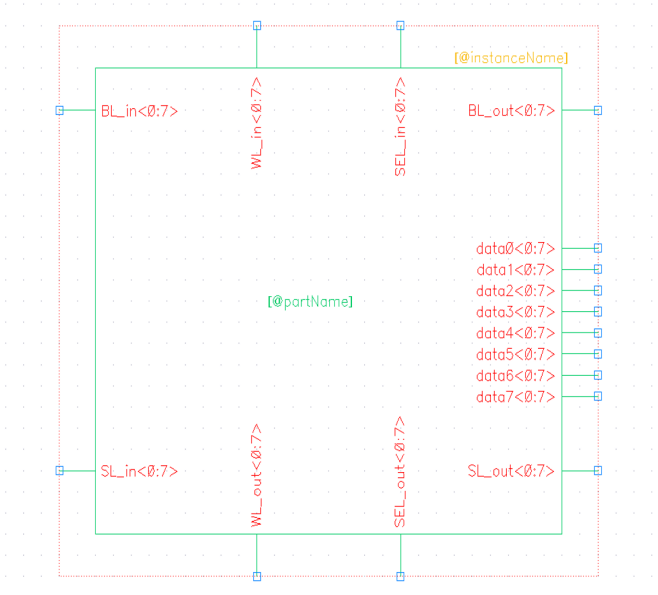


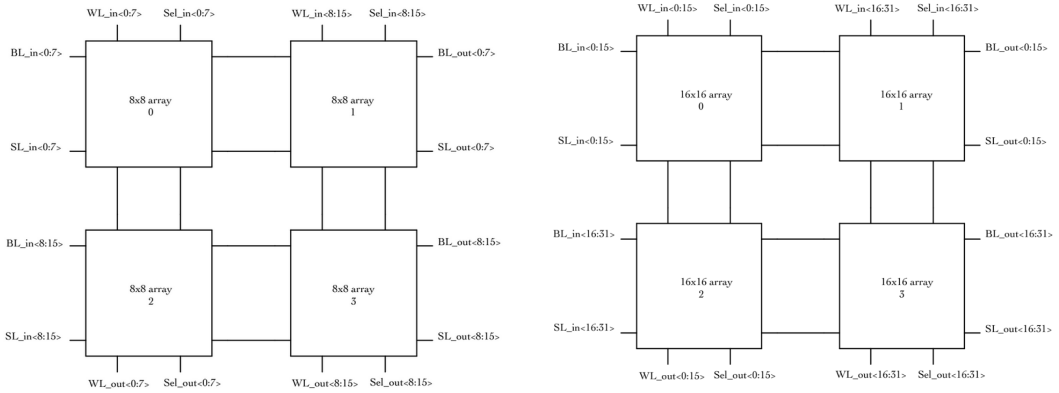
Figure 4.4: Schematic of a 8x8 FeFET–based array.

For what concern the design of the memory array, it has been conducted

at schematic level in Cadence Virtuoso, using the *CMOS 28nm FD–SOI* technological library and a custom library containing the FeFET model. The first array to be implemented is an 8x8 array and its schematic is shown in Figure 4.4. Here, the aforementioned memory word (column) in Figure 4.3 is replicated eight times, defining the following signals —all of these have two *sides*: an input side and an output side, separated by the cascade of parasitics —:

- `BL<0:7>` driven by the sense amplifier peripheral (section 4.3);
- `SL<0:7>`, `WL<0:7>` for write and read purposes (section 4.4);
- `Sel<0:7>` for read purposes (subsection 4.2.4);
- from `data0<0:7>` to `data7<0:7>` for debugging purposes. That is, these signals represents the internal polarization of each cell. These are measurable voltage nodes with unit $\mu C\,cm^{-2}$.

4.2.2 Larger dimension memory array



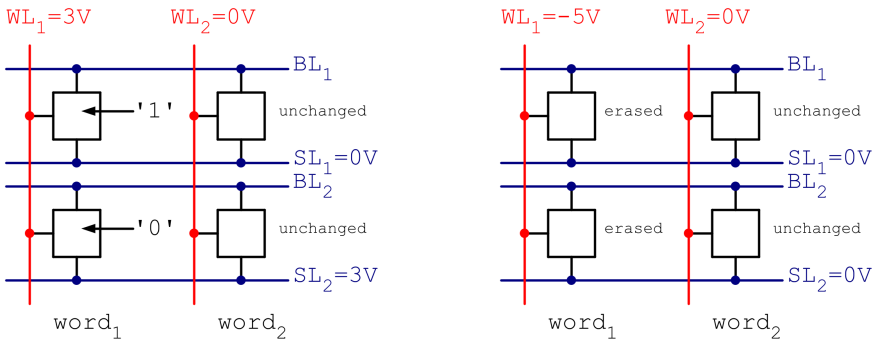
(a) Schematic of a 16x16 FeFET–based array.

(b) Schematic of a 32x32 FeFET–based array.

Figure 4.5: Schematics of larger dimension arrays.

Starting from the 8x8 array shown in subsection 4.2.1, larger arrays can be obtained at a schematic level by appending more than one smaller array. That is, a 16 words array and a 32 words array are shown in Figure 4.5. Here, the signals have the same meaning as described in subsection 4.2.1.

4.2.3 Writing schemes



(a) Programming '10' to the first word.

(b) Erasing the first word.

Figure 4.6: Writing schemes of the FeFET–based memory array.

The writing operation in the proposed memory architecture involves one word (column) per time and no other operation —such as reading — is allowed meanwhile.
Therefore, the writing operation is controlled by the WL and SL signals, while every cell is unplugged from the bit lines during the operation by means of the selector in cut off.
For what concern the simulation in Cadence Virtuoso, ideal PWL generators from the *analogLib* drive both word lines and source lines, as it is explained in section 4.4.
According to the convention in FLASH memories, the writing is distinguished in two operations: *programming* and *erasing*. In the former case, a positive voltage of $V_G = 3\,\mathrm{V}$ on the FeFET gate is sufficient to rise the polarization to the high conduction level. On the other hand, in the latter case, $V_G = -5\,\mathrm{V}$ is required.

Programming scheme

Programming refers to the operation of writing '1' to the desired memory cell inside a word. Thus, the word to be written must contain only zeroes before being programmed.
This process is shown in Figure 4.6(a). Here, the first word is being written. Suppose to write '10':

- the selected word line is driven to 3 V, the others to 0 V, in order to lead the gate of the selected FeFET to the programming voltage;

- the selected source line is driven to 0 V, the others to 3 V in order to set the proper voltage drops on every cell.

Erasing scheme

Erasing refers to the process of deleting the whole content stored in a memory word in order to write the new word to it.
Thus, as depicted in Figure 4.6(b):

- the selected word line is driven to −5 V, the others to 0 V, in order to lead the gate of the selected FeFET to the erasing voltage;

- all the source lines are driven to 0 V.

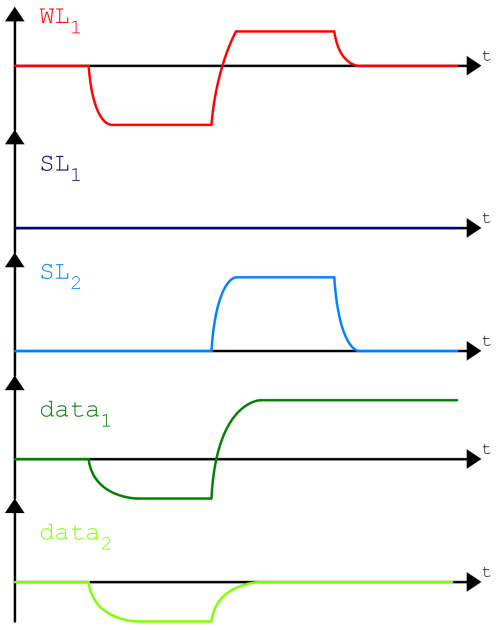


Figure 4.7: An example of writing timing referred to Figure 4.6.

Finally, Figure 4.7 shows an example of a full write operation of the first word as in Figure 4.6. That is, an erasing operation preceeds the programming operation.

4.2.4 Reading schemes

The reading operation involves a pheriperal circuit to convert the current of the cell to a logic voltage output. That is, a sense amplifier capable of reading an entire memory word has been simulated in Cadence Virtuoso and is described in section 4.3.
Moreover, one word per time is read, meaning that the column to be read is plugged to the bit line bus.

Considering the schematic in Figure 4.8, reading a word in the memory

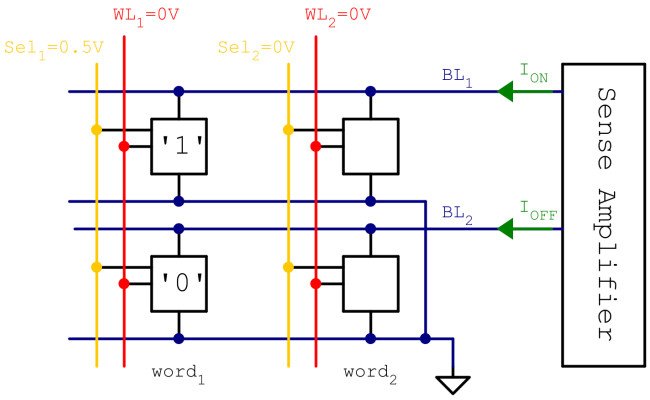


Figure 4.8: Reading scheme of the FeFET–based memory array, involving a Sense Amplifier.

leads to the following operations:

- during a phase of memory initialization, the reference cells inside the sense amplifier must be programmed to '1'. Thus, the dedicated reference word line is used, in particular $\mathtt{WL_{Ref}} = 3\,\mathrm{V}$.
- the selected word (column) is plugged to the bit line bus —each cell to its own bit line— through the select signal, generated by an ideal PWL generator. Thus, $V_{Sel} = 0.5\,\mathrm{V}$ has been chosen to turn on the selector transistor. Reference to Figure 3.11;
- the word line of the selected word is driven to $0\,\mathrm{V}$, as well as all the other words. Nonetheless, the other words are not connected to the bit line bus and cannot alterate the bit line voltage;
- then, a cell can either generate I_{ON} or I_{OFF}, which values depend on the sense amplifier design. This current level is then converted to high or low voltage.

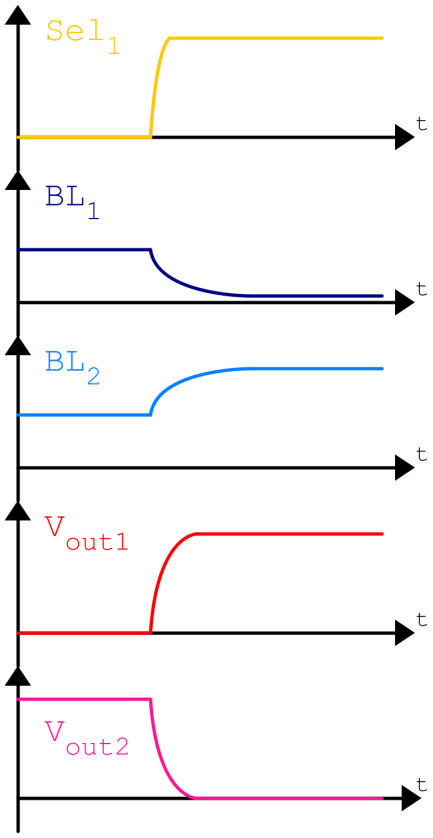


Figure 4.9: An example of reading timing referred to Figure 4.8.

Finally, an example of a read operation performed by the sense amplifier on the first column in Figure 4.8 is presented in Figure 4.9. Here, it is shown how the sense amplifier manages to convert the output current to a voltage variation on the bit line and senses it to generate a voltage output. Therefore, the details of this process are discussed in section 4.3.

4.3 Sense Amplifier

The reference for the design of the sense amplifier is derived from [40] and [41], where topologies for conventional current sense amplifiers for NOR FLASH memories are reported.
In particular, the schematic simulated in Cadence Virtuoso will be described as well as the testbench and the performance measurements.

4.3.1 Schematic

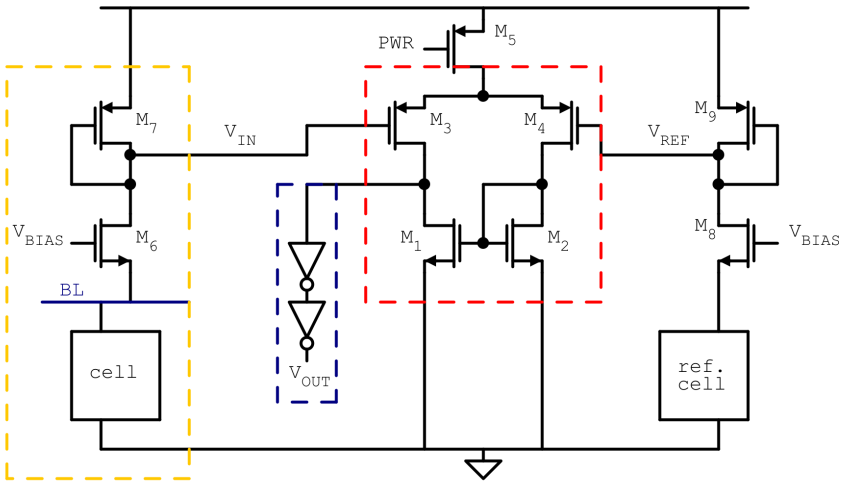


Figure 4.10: Schematic of the sense amplifier. In the yellow box the cascode amplifier; in the red box the differential pair; in the blue box the output buffer.

The schematic of the sense amplifier is shown in Figure 4.10. It is composed by:

- a first cascode stage which amplifies the voltage on the bit line to the input voltage to the second stage. In other words, the current generated by the cell is converted to voltage. The common gate MOSFET is driven by $V_{BIAS} \simeq 0.6\,\mathrm{V}$ only during the reading phase, otherwise it is off.

- a second differential pair stage, a n–channel current mirror, which takes two inputs, V_{IN} from the cell —by the first stage— and V_{Ref} from a dummy cell which is always programmed. V_{Ref} is similarly generated by a cascode stage but the transistors dimensioning is such that this voltage level is fixed and close to half of the sensing dynamic. Moreover, $V_{DD} = 1\,\mathrm{V}$. To cut the power during the non–reading phases, a p–type MOSFET controlled by the PWR signal cuts the current to the current mirror.

- a final output buffer stage, composed by two cascaded CMOS inverters. Its purpose is to adjust the values of V_H and V_L to, respectively, 1 V and 0 V, since the dynamic is reduced due to the transistors overdrives.

Moreover, the dimensioning of the transistors is the following:

$$W_1 = W_2 = W_7 = W_8 = 80\,\text{nm} \tag{4.1}$$
$$W_3 = W_4 = W_5 = W_6 = W_9 = 160\,\text{nm} \tag{4.2}$$

In particular, since $W_9 = 2 \cdot W_7$, the drain voltage V_{IN} is affected by a larger overdrive than the drain voltage V_{Ref} when the current is the same in both branches. That is the case when the cell stores '1' and the voltage on the bit line —and then V_{IN}— is driven below $V_{Ref} \simeq 300\,\text{mV}$. Thus, the differential pair converts this difference to V_H.
Otherwise, when the cell stores '0', a negligible current flows in the input branch, then V_{IN} is above V_{Ref}. The output is so driven to V_L.
For what concerns the dimensioning of the differential pair, $W_3 = W_4 = W_5 = 2 \cdot W_1 = 2 \cdot W_2$ in order to adjust V_H as close as possible to V_{DD}.
Moreover, an enhancement to the conventional schematic in [40] is apported. That is, the output is driven to 0 V by a pull–down transistor when the sense amplifier is unused.
Finally, the schematic level in Cadence Virtuoso is shown in Figure 4.11.

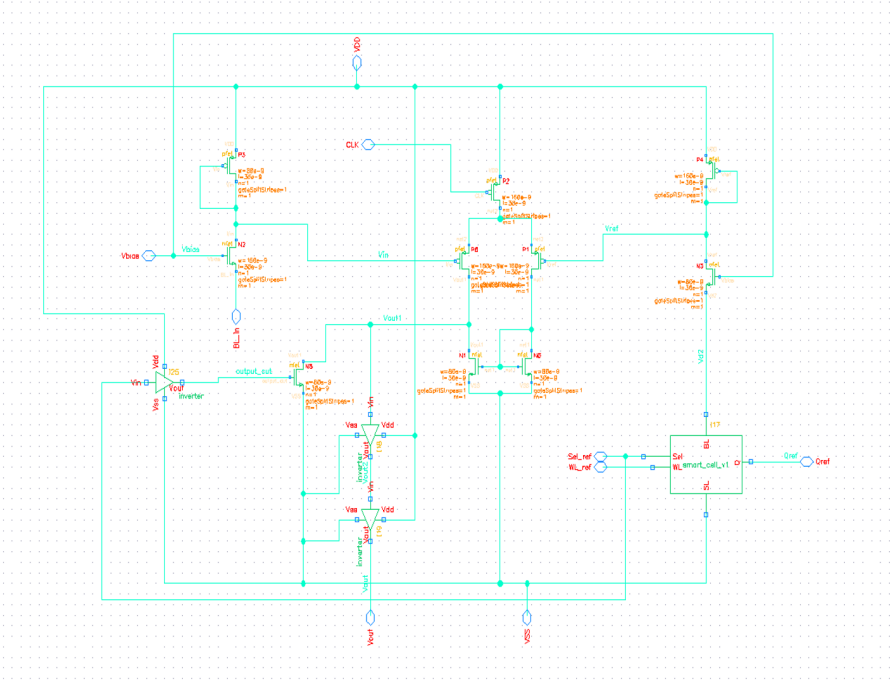
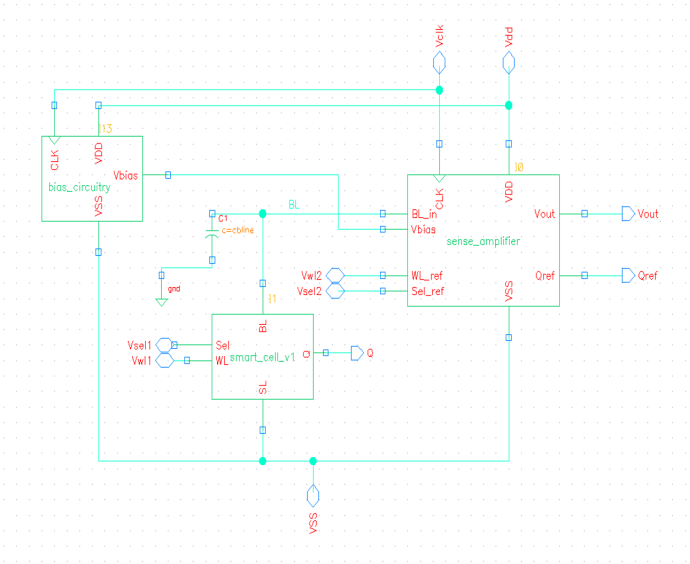
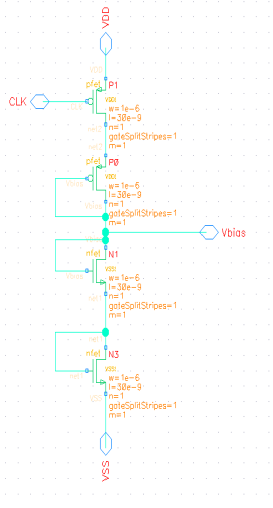


Figure 4.11: Schematic level of the sense amplifier in Cadence Virtuoso.

4.3.2 Testing and performance



(a) Testbench of the sense amplifier.



(b) Scheme of the bias circuit for the sense amplifier.

Figure 4.12: Complete circuit of the testbench of the sense amplifier.

A single instance of the sense amplifier has been tested by means of a single memory cell, where a capacitance has been appended to the bit line in order to emulate the load capacitance of a full array. The testbench is shown in Figure 4.12(a), with a bit line load capacitance of $C_{\text{bit line}} = 100\,\text{fF}$.
Moreover, the bias voltage to drive the sense amplifier is generated by the subcircuit in Figure 4.12(b).

Furthermore, the simulated waveforms in the case of a reading of a cell containing '1' are plotted in Figure 4.13.
In particular, the amplification of the cascode stage results in a speeding–up of the V_{IN} signal with respect to the voltage on the bit line.
Moreover, the non–buffered output is compared to the buffered output.
Finally, the input select signal and the sense amplifier output are compared to show the delay. That is, under these conditions it is almost 150 ps.

As far as the power consumption is regarded, the profile shown in Figure 4.14 exhibits $P_{peak} \simeq 40\,\mu\text{W}$, with an average dynamic power of about $25\,\mu\text{W}$.

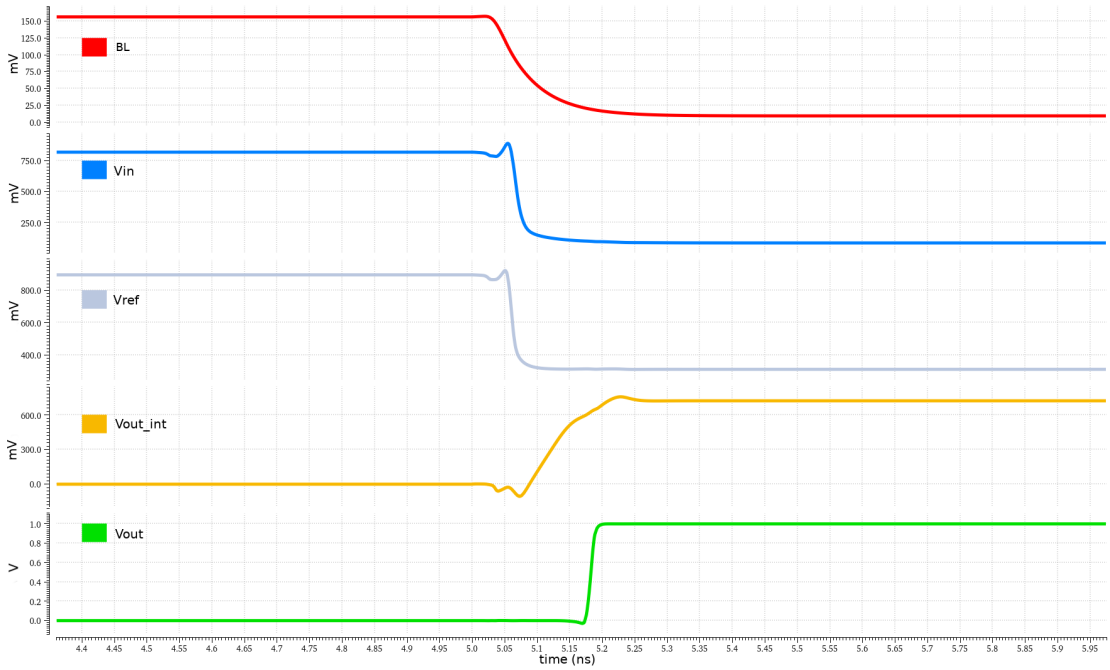
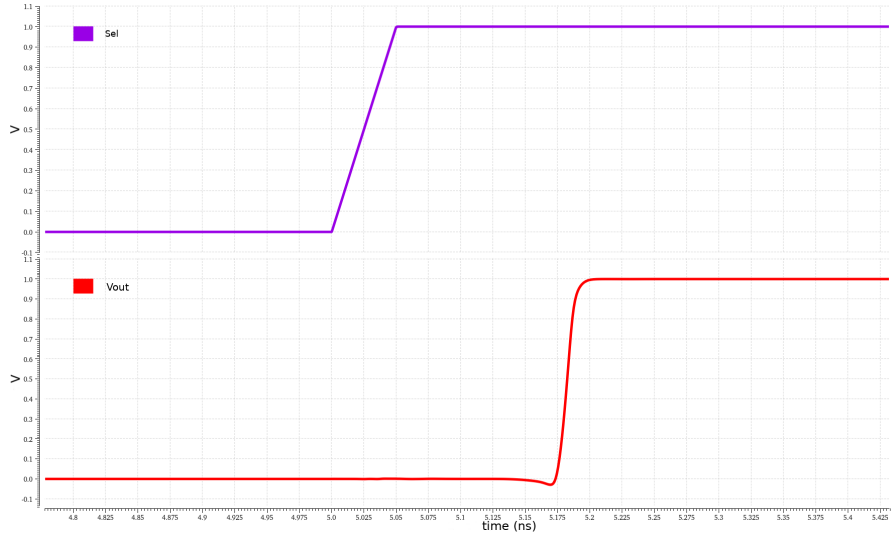


Figure 4.13: Signals in the cascode, differential pair and output stage during the simulation of a reading.

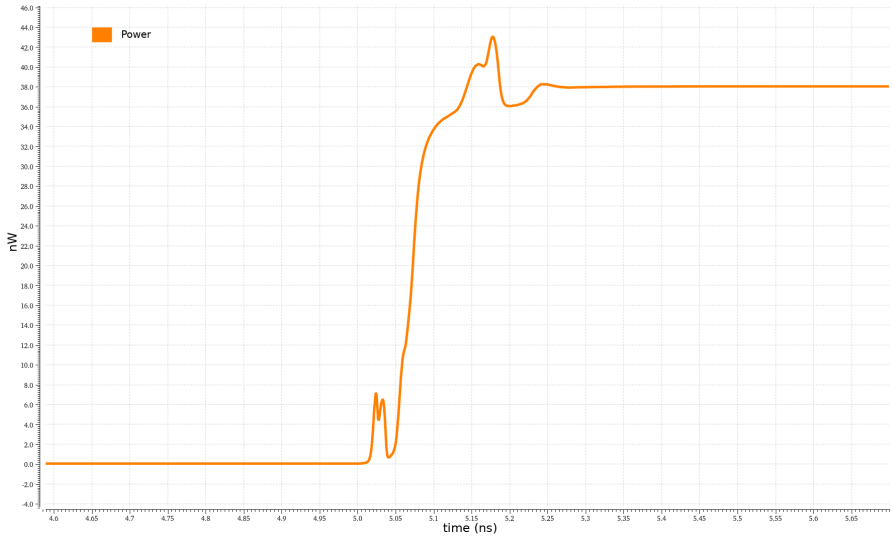
That is a reasonable result if compared to current sense amplifiers in literature such as in [42], [43], [44].

4.4 NOR FeFET memory array: Testbench and scripts

In order to characterize a complete memory array with FeFET–based cells, three versions have been created. That is, N words–N bit arrays with N = 8,16,32.
These are shown in Figure 4.15, where all the driving signals such as WLs, SLs, PWR and Select are generated through ideal PWL generators from the *analogLib* in Cadence Virtuoso. These generators are then written by means of python scripts which is presented in subsection 4.4.1.
Moreover, the testbenches in Figure 4.15 are completed by a pheripheral column–sense amplifier composed by N instances of the single sense amplifier described in section 4.3.
Therefore, the allowed operations inside each memory are:



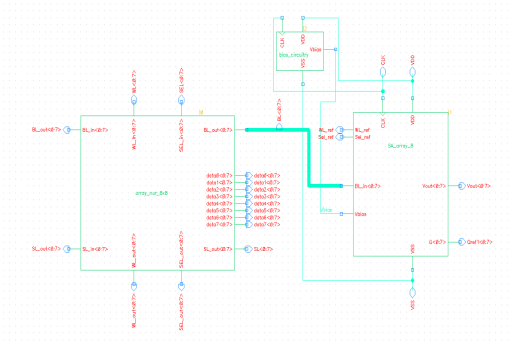
(a) Input selection signal and output voltage comparison.



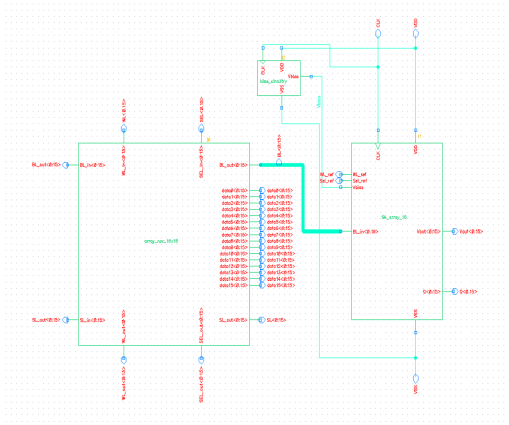
(b) Power profile of the sense amplifier.

Figure 4.14: Waveforms of the simulation of the sense amplifier while the cell contains '1'.

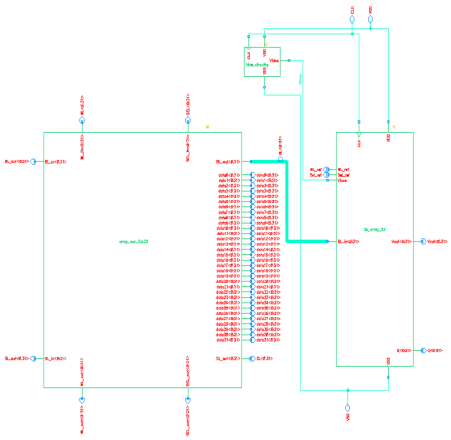
- **reading a word (column)**: in this case, the performance evaluated are the reading delay low to high (LtoH) and high to low (HtoL) during the reading of the cell in the first column and in the last row. In fact, this position is the farthest from both the Select drivers and the sense amplifier (Figure 4.16).



(a) Testbench of the 8x8 array.



(b) Testbench of the 16x16 array.



(c) Testbench of the 32x32 array.

Figure 4.15: Testbench of varying dimension memory arrays.

For what concerns the power dissipation, the average power consumption is derived for both the array and the sense amplifier. Here, the measurements refer to the reading of the entire word.

- **erasing a word (column)**: in this case, the erasing delay is calculated considering the first column and the last row due to the distance from the WL drivers. Since the sense amplifier is off, only the average power consumption for the array is taken. In particular, the entire word containing '111...1' is erased;

- **programming a word (column)**: in this case, the programming delay

is calculated considering the same motivations as above. Regarding the average power, same considerations as above except by the fact that only one '1' is written in the word. Thus, the maximum allowed number of source lines switches, being more power demanding;

- **idle**: during an idle phase in which the memory is unused, the average power dissipation is calculated.

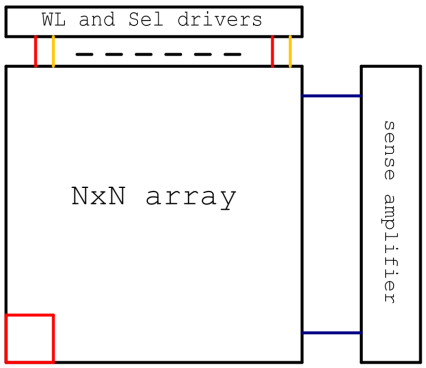


Figure 4.16: Worst case cell for reading and writing performance.

4.4.1 Python scripts

Every PWL generator from the *analogLib* in Cadence Virtuoso takes a *.csv* input file to generate a custom waveform. Also, the input file has the following format: `<time> <value>`.
Therefore, two Python scripts are created to manage, respectively, the generators of the array —`WL`, `SL`, `Select`— and the signals fo the sense amplifier. In fact, the array may require a quite large number of `.csv` file, which are time demanding to handle without a script.

Array signals

The script for the array signals contains a functions whose arguments are:

- **operation**: erasing, programming or reading;
- **cycle**: during which cycle the operation happens;

- **duration**: how long, in total, the simulation is going to last;
- **word**: the word to be written to the memory in case of programming. Otherwise, it is unused;
- **col**: at which address (column) the operation is taking place;
- **time_offset**: if different from 0, allows to append further operations to the simulation.

Moreover, the script specifies some parameters such as the duration in seconds of a cycle, the array dimension, the rise/falling time and V_{DD}.

```
            # WL now
            for wl in range(len_word):
                if wl==col and t==cycle:
                    file_WL[wl].write(str((t-1)*t0 + t_rise) + " " + '3' + "\n")
                    file_WL[wl].write(str(t*t0) + " " + '3' + "\n")
                else:
                    file_WL[wl].write(str((t-1)*t0 + t_rise) + " " + '0' + "\n")
                    file_WL[wl].write(str(t*t0) + " " + '0' + "\n")
```

Listing 4.1: Source code of the Python script for the array signals. The generation of the word lines is shown.

In the source code above, it is shown how the word lines are generated in case of a programming operation. Thereby, the script sweeps all the word lines in the memory, for each cycle out of the total duration, and writes 3 V to the correct memory address (column) at the correct cycle.

```
            # SL now
            for sl in range(len_word):
                if t==cycle:
                    if int(word[sl])== 0:
                        file_SL[sl].write(str((t-1)*t0 + t_rise) + " " + '3' + "\n")
                        file_SL[sl].write(str(t*t0) + " " + '3' + "\n")
                    elif int(word[sl])== 1:
                        file_SL[sl].write(str((t-1)*t0 + t_rise) + " " + '0' + "\n")
```

```
9                           file_SL[sl].write(str(t*t0) + " " + '
    0' + "\n")
10                  else:
11                      file_SL[sl].write(str((t-1)*t0 + t_rise)
    + " " + '0' + "\n")
12                      file_SL[sl].write(str(t*t0) + " " + '0' +
     "\n")
```

Listing 4.2: Source code of the Python script for the array signals. The generation of the source lines is shown.

Here, instead, the generation of the source lines in case of programming is shown. As discussed in subsection 4.2.3, the source lines are sweeped and 0 V is written if a '1' is addressed to that cell and the cycle is correct, 3 V otherwise.
Furthermore, the other operations behave in a similar way. The entire code is left to the Appendix.

Sense Amplifier signals

The script for the sense amplifier signals contains a functions whose arguments are:

- **signal**: the signal to be controlled. Each signal is associated to an operation (e.g. $\texttt{WL}_{\texttt{Ref}}$ is associated to programming the dummy cell during the initialization of the memory);
- **start**: the starting cycle of the operation associated to the signal;
- **stop**: the final cycle of the operation associated to the signal;
- **duration**: how long, in total, the simulation is going to last;
- **time_offset**: if different from 0, allows to append further operations to the simulation.

Therefore, the code has the same structure of the code portions shown previously.

4.5 NOR FeFET memory array: measurements and performances

As explained in section 4.4, several measurements are taken to characterize the proposed memory array. Therefore, in the following, the main waveforms

will be shown as long as graphs to put in comparison the arrays of different dimensions.
Moreover, the writing and reading operations follow the concepts discussed in subsection 4.2.3 and subsection 4.2.4, where the sequence of necessary steps are shown in Figure 4.7 and Figure 4.9. Therefore, Figure 4.17 shows a

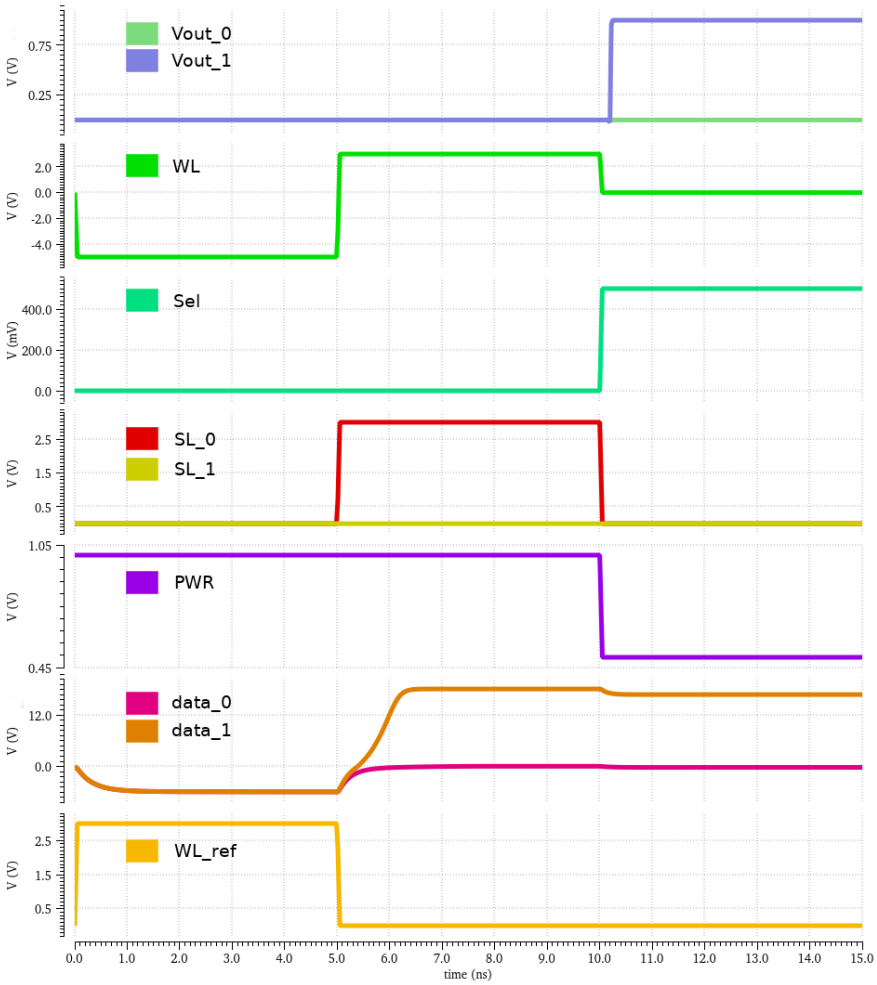


Figure 4.17: Simulation of memory operations sequence in three cycles: erasing, programming and reading. 8x8 array.

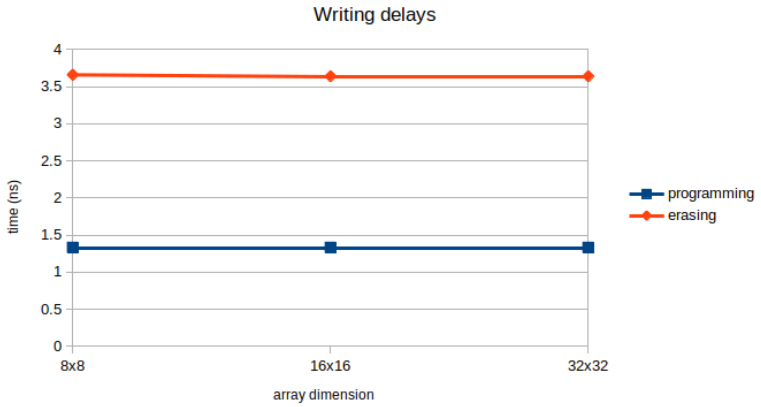
simulation of three cycles of memory operations. It is shown how the signals manage the three allowed operation of the array.

Parameter	value
V_{DD}	1 V
t_{rise}, t_{fall}	50 ps
t_0 (period)	5 ns
t_{FE} (thickness)	100 nm
V_{read}	0 V
V_{select}	500 mV
$V_{program}$	3 V
V_{erase}	−5 V
Integration period	1 ns*, 5 ns**
Thresholds for signal delay	90%–90%

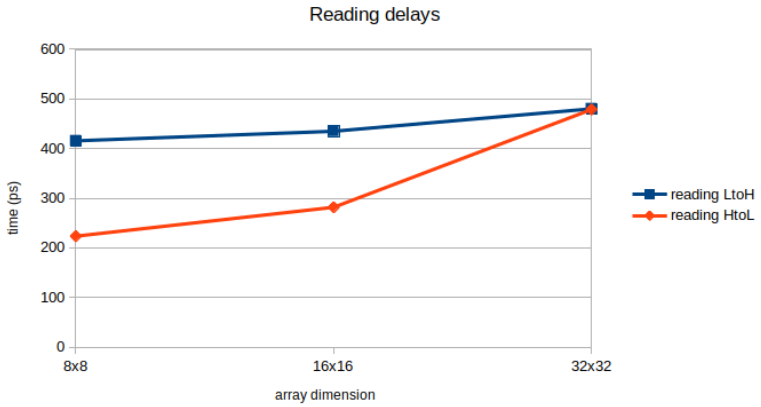
Table 4.1: List of the simulation parameters used for the performance measurments.
*: integration period for the power during a read operation;
**: integration time for the power during a write operation.

4.5.1 Delays

The extracted delays are summarized in Figure 4.18. Those results are obtained considering an input slew —the rising and falling time of the signals generated— of 50 ps.
The programming operation appears to be faster than the erasing operation, due to asymmetries in the FeFET hysteresis, as it is clear from Figure 3.8 and Figure 3.9, for instance. In fact, in the tested FeFET model, less efforts in terms of voltage level and duration of the pulse are required to switch on the polarization than to switch it off.
For what concern the reading operation, the results refer to the performance of the sense amplifier already introduced in section 4.3. Here, the reading of a cell containing '1' tends to be slower than the other case. Furthermore, in Figure 4.19 the waveforms of the word line and the data inside the cell —the polarization of the FeFET— are shown to have a visual example of the process inside the cell.
In Figure 4.20 the waveforms of the select line and the output of the sense amplifier are shown to demonstrate the behavior of the device applied to an array.



(a) Reading delays.

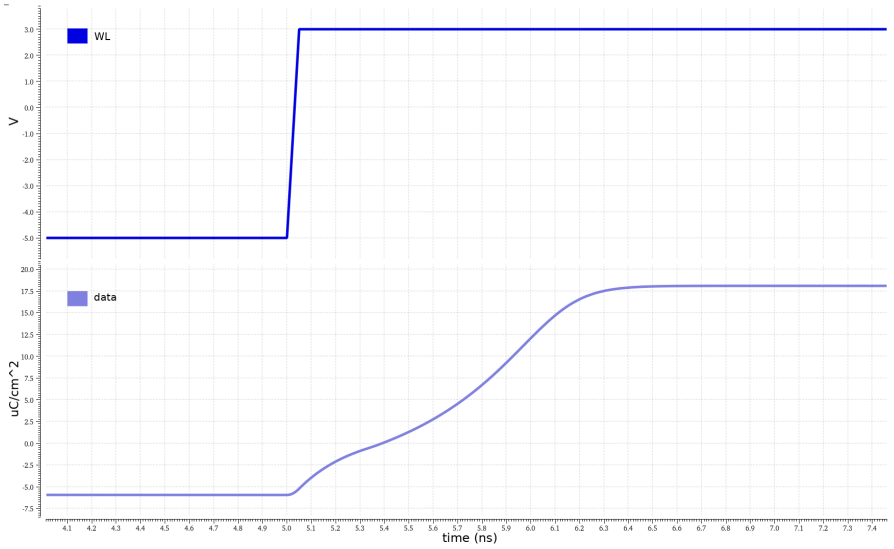


(b) Writing delays.

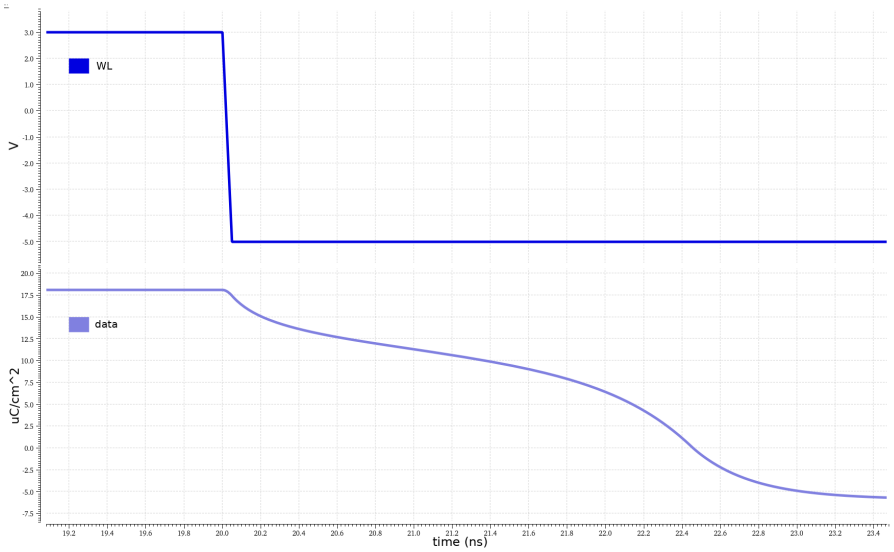
Figure 4.18: Graphs of the simulated delays of the memory arrays.

4.5.2 Power consumption

The extracted average power of the array and the sense amplifier are summarized in Figure 4.21, Figure 4.22 and in Figure 4.23. Since the operations of writing and reading are affected by different delays, periods of 5 ns and 1 ns are used, respectively, to integrate the waveforms. Moreover, the idle power is integrated over 5 ns.
Therefore, the programming dynamic power is higher than the erasing power, since the FeFET, when programmed to the high conduction state, might be affected by current leakage which leads to greater peak currents as shown in Figure 4.24.
Moreover, in Figure 4.24, negative peaks are generated at the end of the



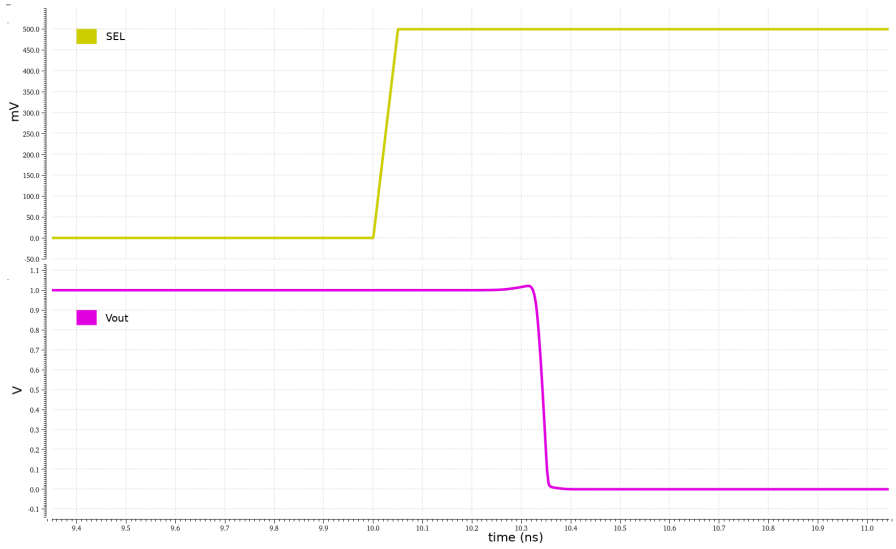
(a) Program delay.



(b) Erase delay.

Figure 4.19: Programming and erasing delays in the 32x32 array.

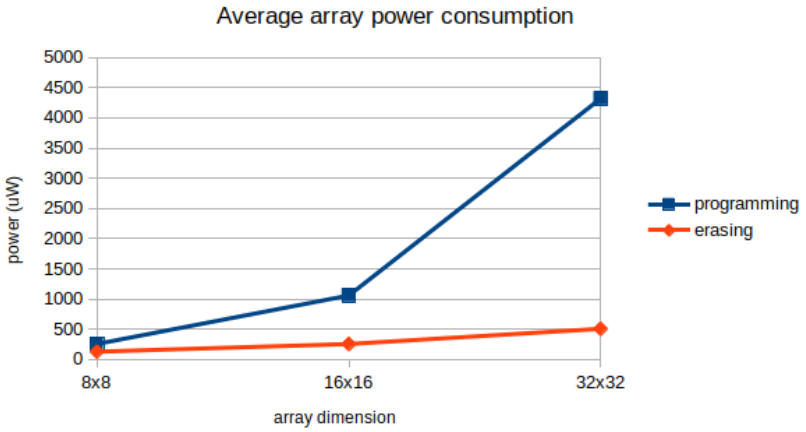
writing cycle. This behavior is connected to the ideal generators going back to an idle state, where the power profiles is calculated by means of a sum of all the current–voltage products of the generators. In those points, the current tends to switch its verse.
Also, the array consumes more when a '1' is read, since in that case I_{ON} flows from the bit lines to the source lines.



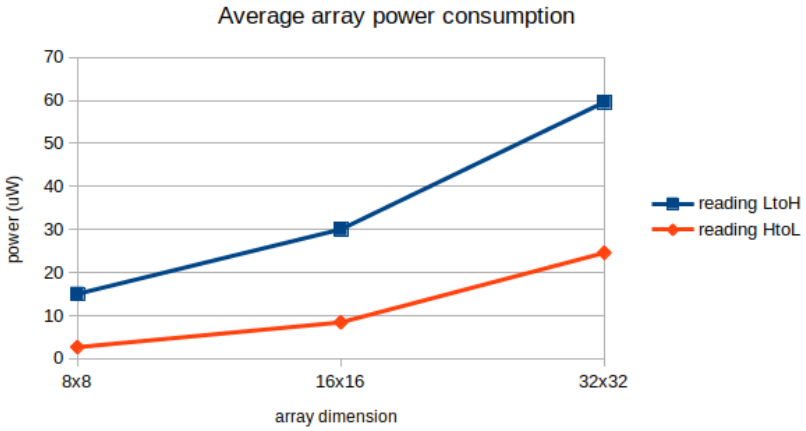
(a) Reading '0' delay.

(b) Reading '1' delay.

Figure 4.20: Reading delays of the sense amplifier for the 32x32 array.



(a) Writing average power consumption.



(b) Reading average power consumption.

Figure 4.21: Graphs of the simulated average power consumed by the arrays during the writing and reading operations.

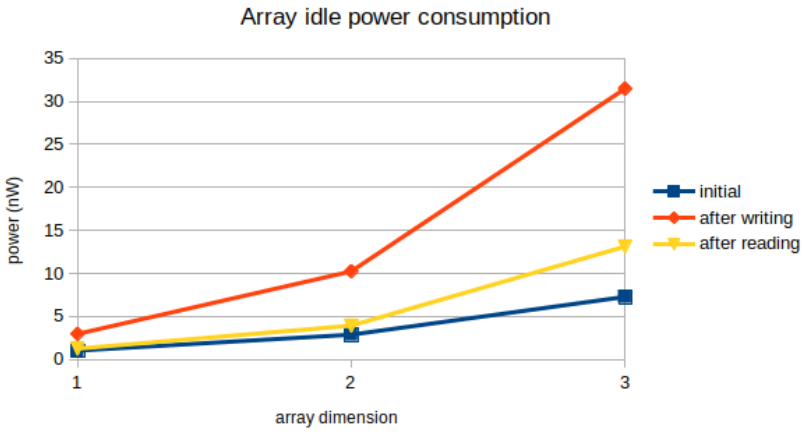
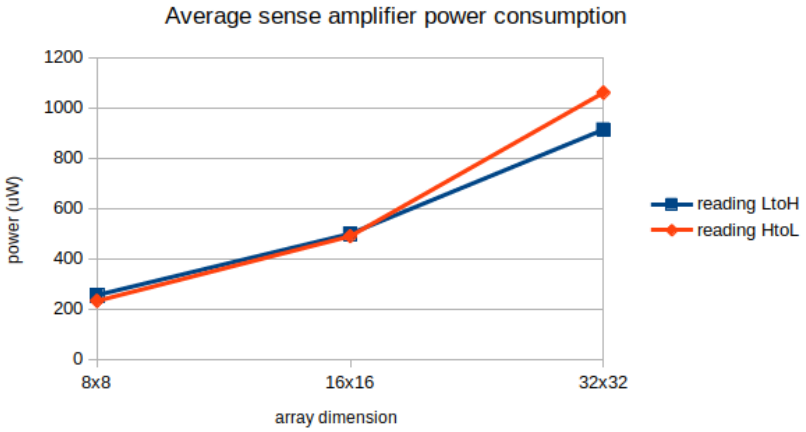
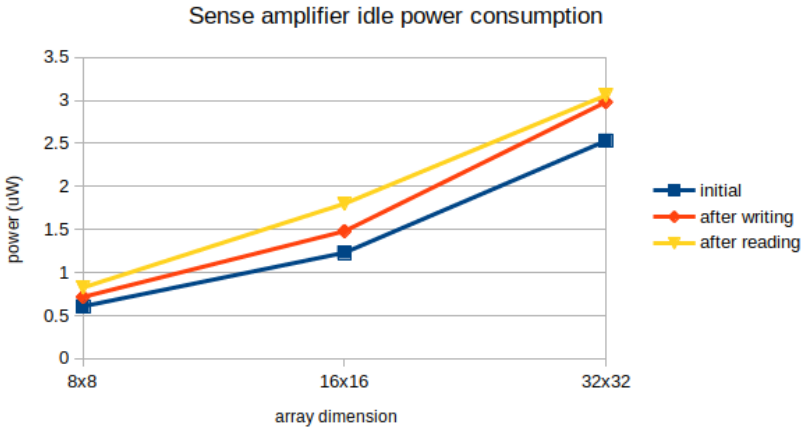


Figure 4.22: Graph of the simulated average idle power consumed by the array during various phases.

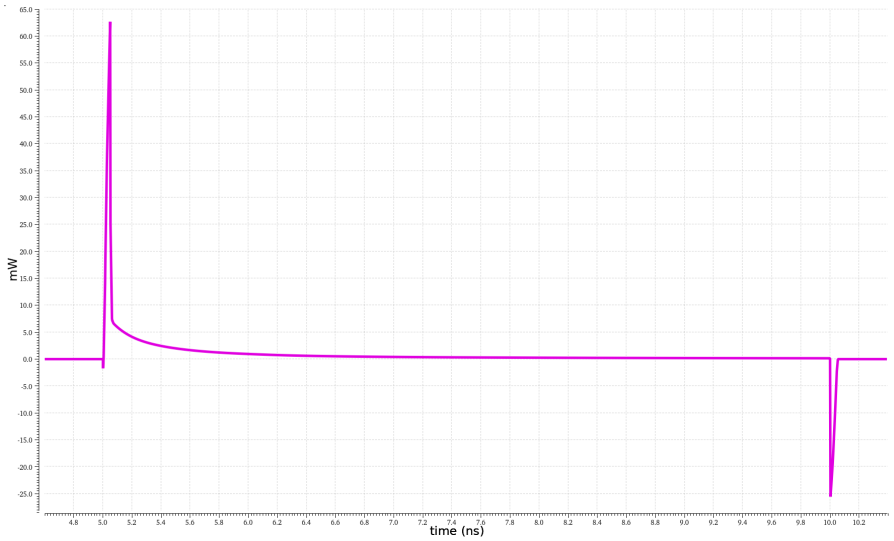


(a) Average power consumption for a reading.

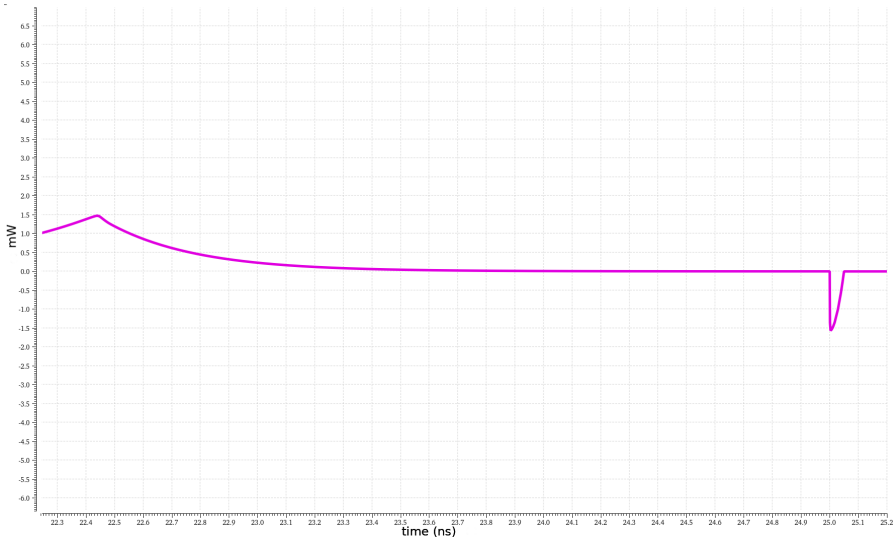


(b) Average idle power consumption.

Figure 4.23: Graphs of the simulated average power consumed by the sense amplifier during the reading operation and in idle.



(a) Power profile during a program operation.



(b) Power profile during an erase operation.

Figure 4.24: Power waveforms during writing operations in the 32x32 array.

Chapter 5

Programmable FeFET–based Logic in Memory

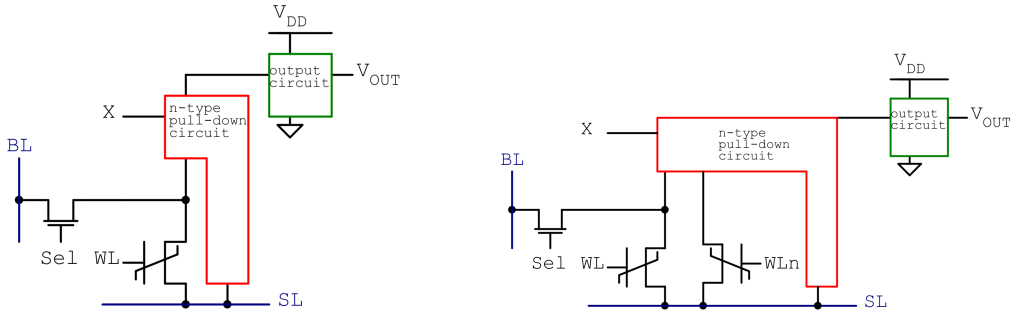
5.1 Overview

The literature presented in subsection 2.2.2 provides several examples of circuits for computing inside the memory. Therefore, the discussed topologies for FeFET–based computing allow to derive solutions for the implementation of smart cells, capable of both memory and logic, which are described in section 5.3. Then, the concept of Liberty file is introduced in section 5.4 and the aforementioned LiM cells are simulated in Cadence Virtuoso and characterized by the Liberty approach.

5.2 LiM: template

5.2.1 Schematic

Every LiM cell presented in this work follows the same template, inspired by Figure 2.40 and discussed in [27].
Therefore, the only variation to take into account is the presence or not of a complementary couple of FeFETs, which store both the data and its inverted value.
In particular, a FeFET–based LiM cell is composed by three main parts and



(a) LiM template of single FeFET cell.

(b) LiM template of double FeFET cell.

Figure 5.1: Schematic of the LiM cell template.

is depicted in Figure 5.1:

- **memory**: one or two FeFETs store the data, similarly to the basic cell discussed in section 3.4. In fact, in case of single FeFET, it is connected to a source line, a word line and a bit line through a selector transistor. Otherwise, if there are two FeFETs, the one storing the original data is connected to the bit line.
  Moreover, the presence of the complementary data would allow the use of a differential sensing circuit, but the design and simulation of a new differential sense amplifier go beyond the aim of this work;

- **n–type logic circuit**: this part is responsible for the logic computation. In particular, a pull–down circuit, which is different according to the typology of LiM cell, generates a current output. Moreover, this part can be composed by standard n–type MOSFETs either stand alone or in combination with FeFETs. In the latter case, FeFETs act as the programmable part of the circuit, allowing the LiM cell to be customizable with more than one logic function.
  The external input(s) `X` is a logic signal which drives the gate of the transistors mentioned above;

- **output circuit**: an output circuit is needed to convert the current output to a logic voltage level. Thus, the schematic in Figure 5.2 is adopted, which is based on the concept of pulling–up the voltage of a `match line` with a diode–connected active load. Moreover, a `PWR` transistor cuts the static power when the logic is unused. Finally, the output is buffered by a CMOS inverter stage.

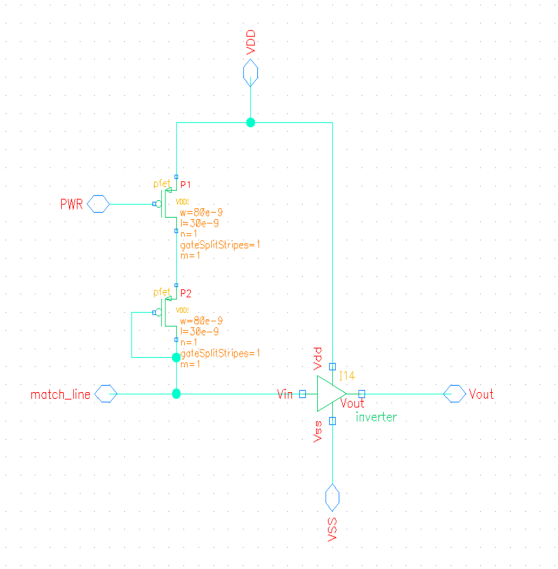


Figure 5.2: Schematic in Cadence Virtuoso of the output circuit of the LiM template.

5.2.2 Testbench

The testbench used for all the instances of LiM cell is shown in Figure 5.3. For the writing operation, `PWL` generators have been used to drive the word lines; for the reading operation, an instance of a sense amplifier with a bit line capacitance of 100 fF is set up; for the logic operation, `PWL` generators are used to drive the external inputs and control signals. Moreover, a load capacitance is added, whose value is parameteric during the simulations as it is discussed in section 5.4.
All the simulations parameters are summed–up in Table 5.1.
Finally, there is a singularity in the following simulations, since writing the cell is executed togheter with the logic operation. Normally, this is forbidden due to the fact that if overlapped, these operations could lead to glitches in the output. Nevertheless, these simulations can be considered *debugging* operations in order to show the correct behavior of the logic circuit.

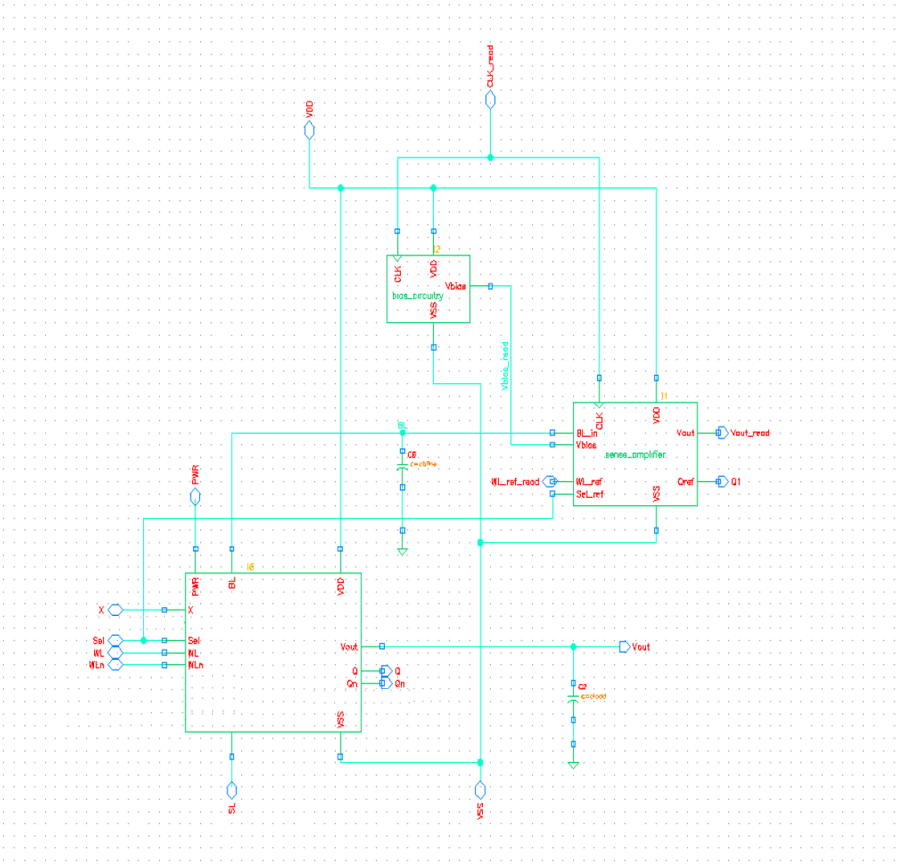


Figure 5.3: Testbench of a generic LiM cell.

5.3 LiM: circuits for computing

The schematic of six LiM cells are presented and simulations are performed to verify the behavior of the cell during the logic operation(s).
Nonetheless, since the writing and reading operations are identical to what presented in subsection 4.2.3 and subsection 4.2.4, these are not discussed here.

5.3.1 LUT cell

Schematic

The LUT cell, whose schematic is shown in Figure 5.4, is a LiM memory cell capable of executing all the possible logic functions with two inputs, by means of 15 transistors. Its logic circuit is made by four branches which are

Parameter	**value**
V_{DD}	1 V
t_{rise}, t_{fall}	$0.01 \cdot t_0$*, varying**
t_0 (period)	40 ns*, 80 ns**
t_{FE} (thickness)	100 nm
V_{read}	0 V
V_{select}	500 mV
$V_H^{ext\_input}$	1 V
$V_{program}$	3 V
V_{erase}	−5 V
Power integration period	t_{rise}, t_{fall} (dynamic) t_0 (static)
Thresholds for signal delay	50%–50%
Thresholds for rise and fall time	30%–70%
Bit line capacitance	100 fF
Load capacitance	100 fF*, varying**

Table 5.1: List of the simulation parameters used for the performance measurments of the LiM cells.
*: value for the LiM simulations;
**: value for the Liberty characterization (section 5.4).

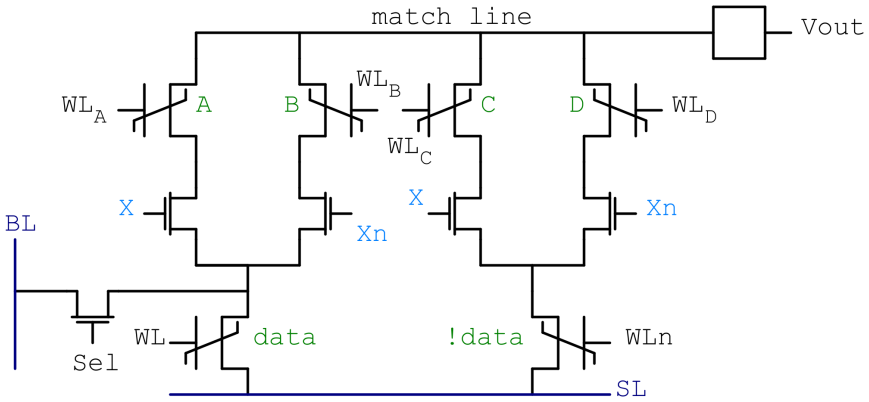


Figure 5.4: Schematic level of the LiM LUT cell.

provided by a programmable FeFET — `A`, `B`, `C`, or `D` —. In each branch, the conduction is activated only by the correct combination of `X` and `data`. Therefore, only one branch conducts per time, and the output value depends

on whether the programmable FeFET is previously written with '0' or '1'. The design of the LUT cell is derived from the Look Up Tables discussed in [26].

Word line(s)	differential
External input number	1
Input line(s)	differential
Logic functions	14
Transistor number	15

Table 5.2: LUT cell features.

Simulation

`data`	`X`	$f(X, data)$
0	0	D
0	1	C
1	0	B
1	1	A

Table 5.3: Truth table of the LUT cell.

Referring to Table 5.3, the four programmable FeFETs can be written to emulate the desired logic function. Thus, the LiM LUT cell is simulated in order to perform both a XOR function and a NAND funtions. Then, three operations are executed: set up the LiM function, writing the cell and the logic operation.
The resultant waveforms are shown in Figure 5.5, Figure 5.6 for the XOR case, and in Figure 5.7, Figure 5.8 for the NAND case.

5.3.2 AND–OR cell

Schematic

The schematic of the AND–OR cell is shown in Figure 5.9. Here is presented a LiM cell which can perform both the AND and OR functions with a total of 11 transistors and 3 programmable FeFETs to select the function. That

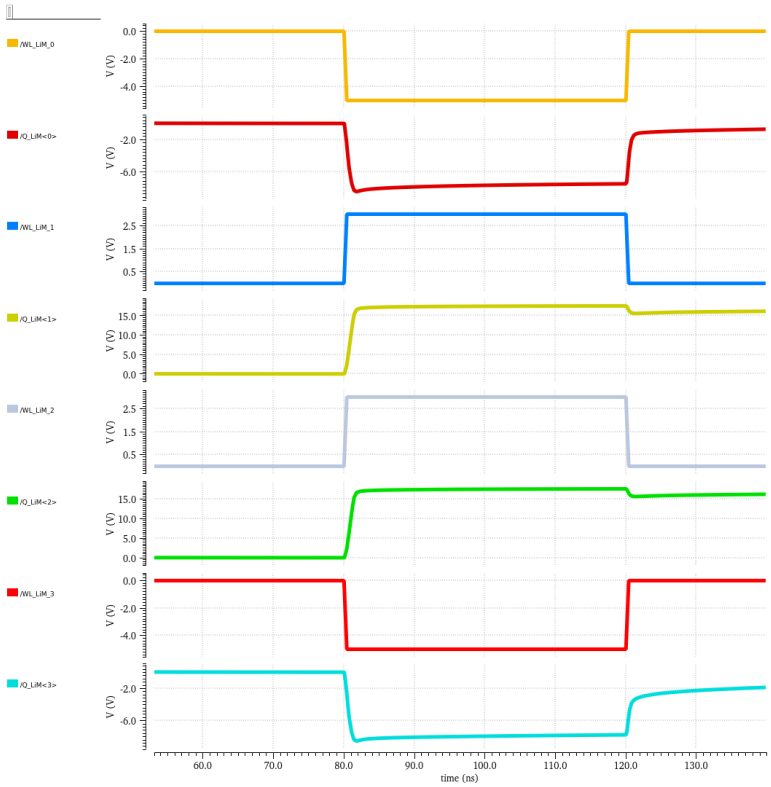


Figure 5.5: Setting up the LiM function by driving the four programming word lines. Simulation of the XOR function programmed to the LUT cell (1).

is, there are two paths in the logic circuit, and only one path per time is activated by properly writing the FeFETs `A` and `B`.

Word line(s)	single
External input number	1
Input line(s)	single
Logic functions	2
Transistor number	11

Table 5.4: AND–OR cell features.

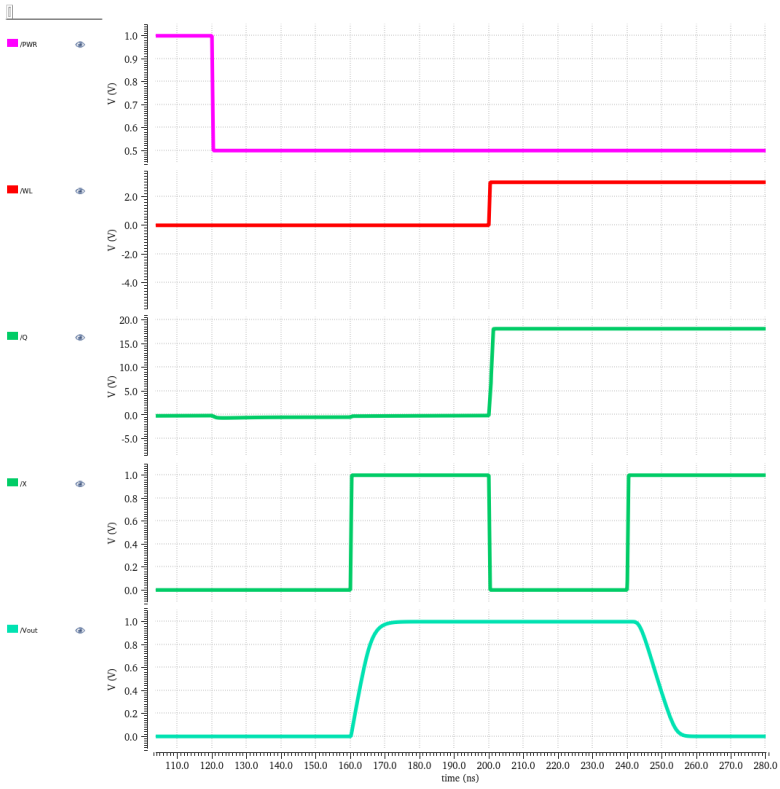


Figure 5.6: Executing the LiM function by changing `data` and `X`. Simulation of the XOR function programmed to the LUT cell (2).

`A`	`B`	$f(A, B)$
0	0	unconditional 0
0	1	OR
1	0	AND
1	1	`X`

Table 5.5: Truth table of the AND–OR cell.

Simulation

In Table 5.5 it is discussed what function the AND–OR cell is performing according to the values written to its programmable FeFETs. Therefore, out of the two cases of interest, the other functions are a `'don't care'`.
Thus, the first part of the simulation regards setting up the cell, while the second part is the logic operation.
The resultant waveforms are shown in Figure 5.10, Figure 5.11 for the AND

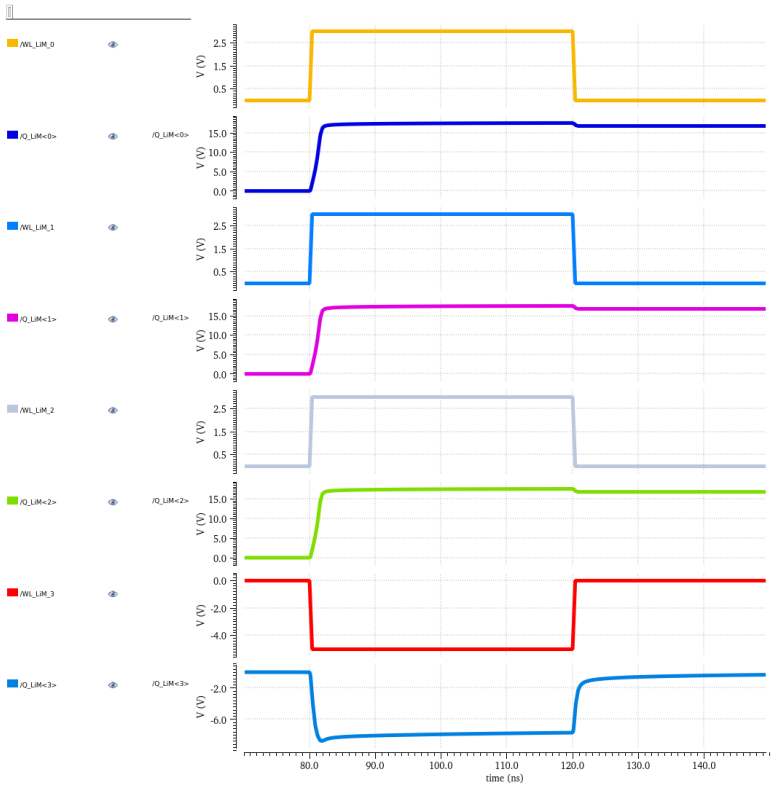


Figure 5.7: Setting up the LiM function by driving the four programming word lines. Simulation of the NAND function programmed to the LUT cell (1).

case, and in Figure 5.12, Figure 5.13 for the OR case.

5.3.3 3–functions cell

Schematic

Figure 5.14 shows the schematic of the 3–functions cell, capable fo executing the three logic functions NOR, AND and XNOR, due to the fact that two programmable transistors enable three different paths in the logic circuit of the cell.
Moreover, this cell has 11 transistors.

Simulation

The logic function allowed by this cell are summarized in Table 5.7.
Following, the waveforms for the simulation for the three possible functions

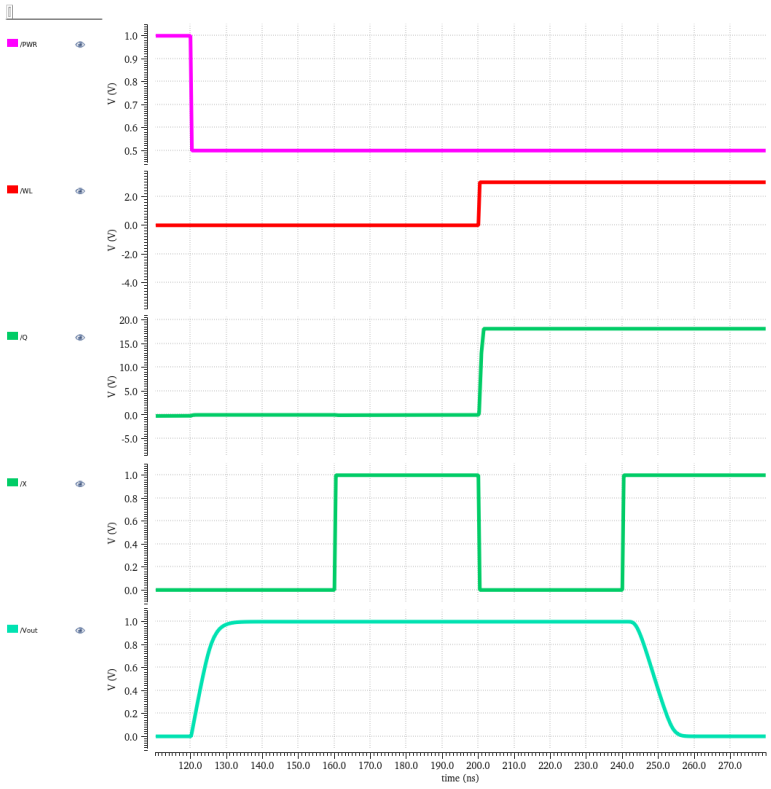


Figure 5.8: Executing the LiM function by changing `data` and `X`. Simulation of the NAND function programmed to the LUT cell (2).

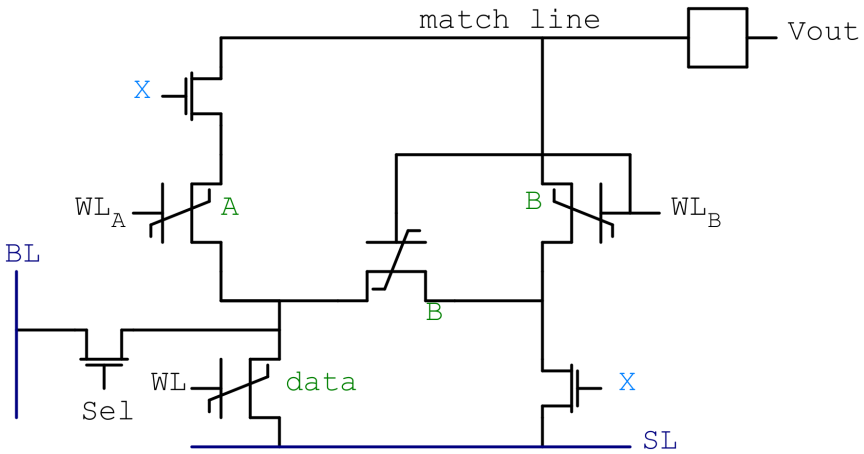


Figure 5.9: Schematic level of the LiM AND–OR cell.

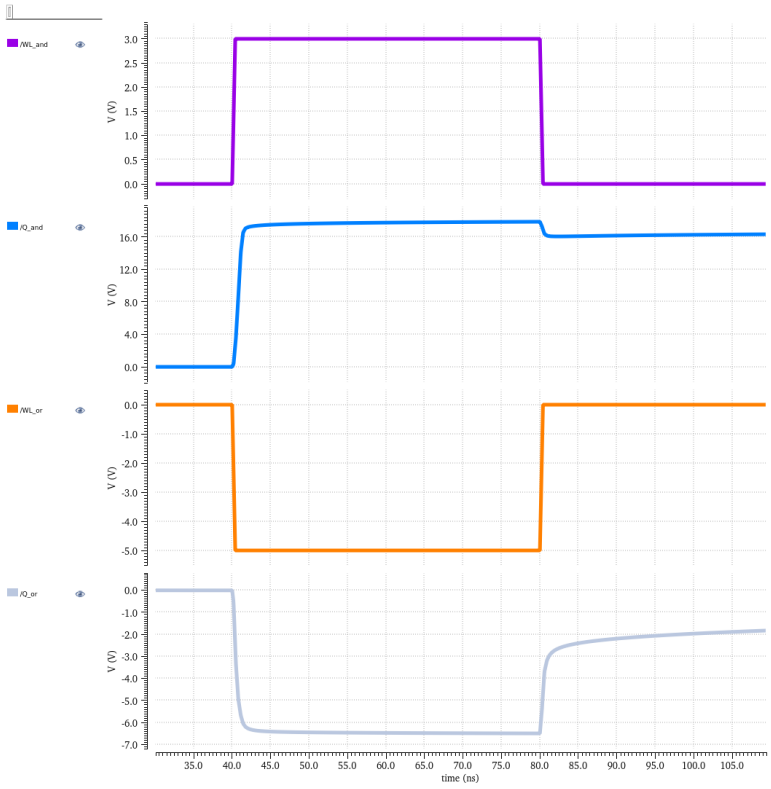


Figure 5.10: Setting up the LiM function by driving the two programming word lines. Simulation of the AND function programmed to the AND–OR cell (1).

Word line(s)	differential
External input number	1
Input line(s)	differential
Logic functions	3
Transistor number	11

Table 5.6: 3–functions cell features.

are shown. That is:

- NOR: Figure 5.15 and Figure 5.16;
- AND: Figure 5.17 and Figure 5.18;
- XNOR: Figure 5.19 and Figure 5.20.

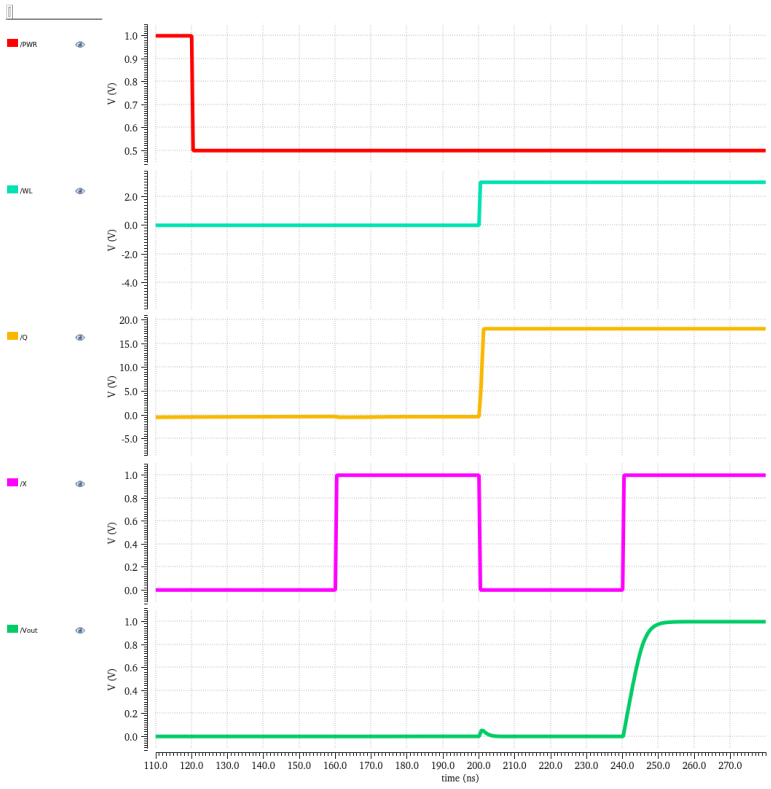


Figure 5.11: Executing the LiM function by changing `data` and `X`. Simulation of the AND function programmed to the AND–OR cell (2).

A	B	$f(A, B)$
0	0	unconditional 0
0	1	NOR
1	0	AND
1	1	XNOR

Table 5.7: Truth table of the 3–functions cell.

5.3.4 XOR cell

Schematic

Contrarily to the LiM LUT, AND–OR and 3–functions cells, the XOR cell, whose schematic is depicted in Figure 5.21, is a non–programmable LiM cell which executes a bitwise XOR between the stored data and an external input. Moreover, the data is stored in a complementary way, and the external input

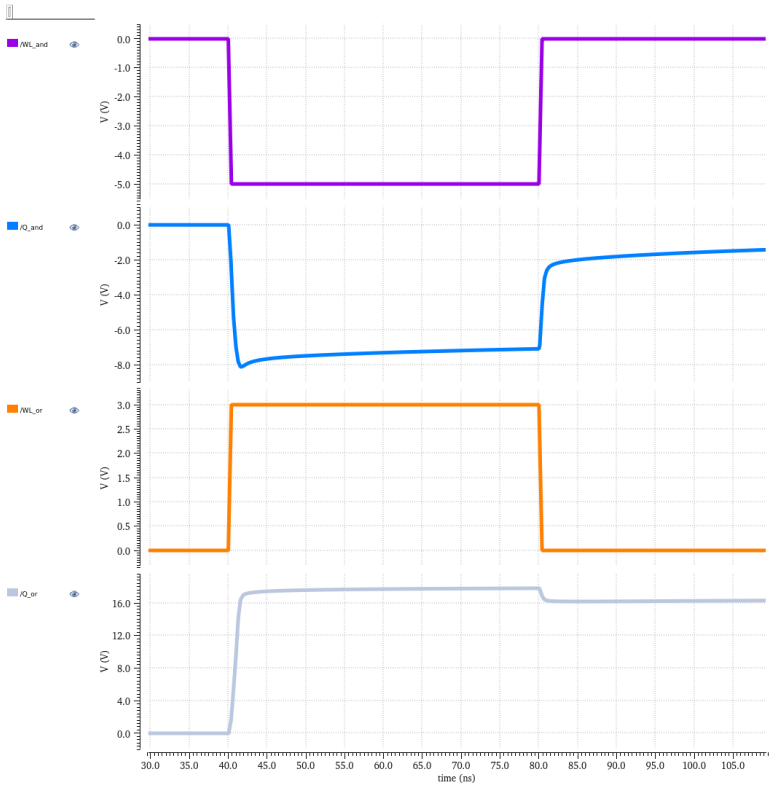


Figure 5.12: Setting up the LiM function by driving the two programming word lines. Simulation of the OR function programmed to the AND–OR cell (1).

is differential, as well.
The logic circuit is composed by two branches which emulate the behavior of the XOR function as a sum of products. This cell has 9 transistors.

Word line(s)	differential
External input number	1
Input line(s)	differential
Logic functions	1
Transistor number	9

Table 5.8: XOR cell features.

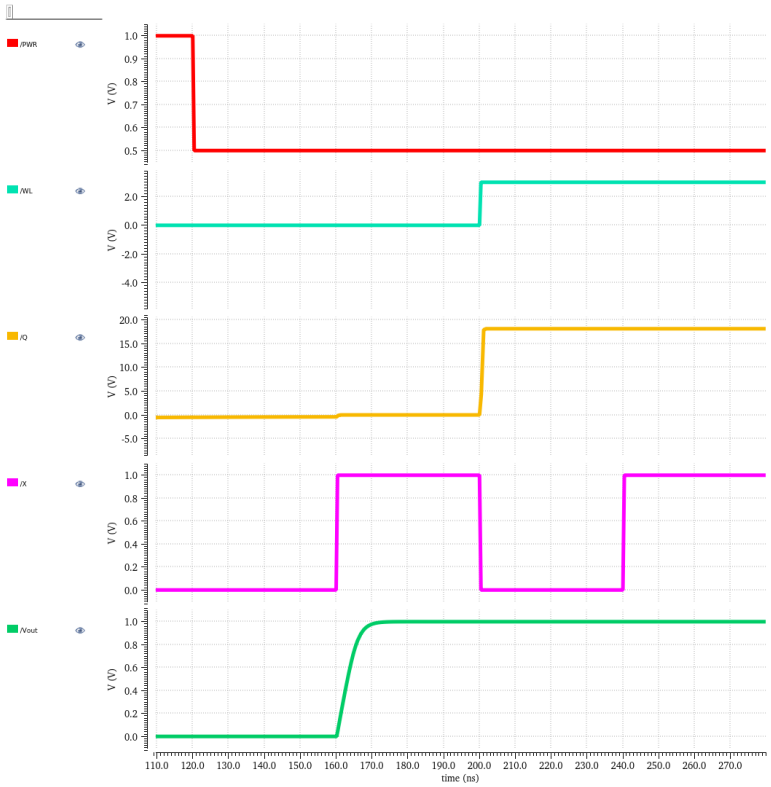


Figure 5.13: Executing the LiM function by changing `data` and `X`. Simulation of the OR function programmed to the AND–OR cell (2).

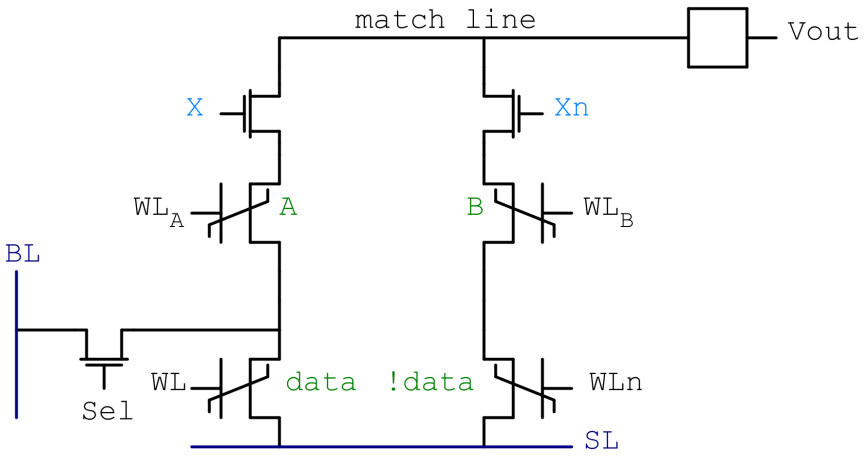


Figure 5.14: Schematic level of the LiM 3–functions cell.

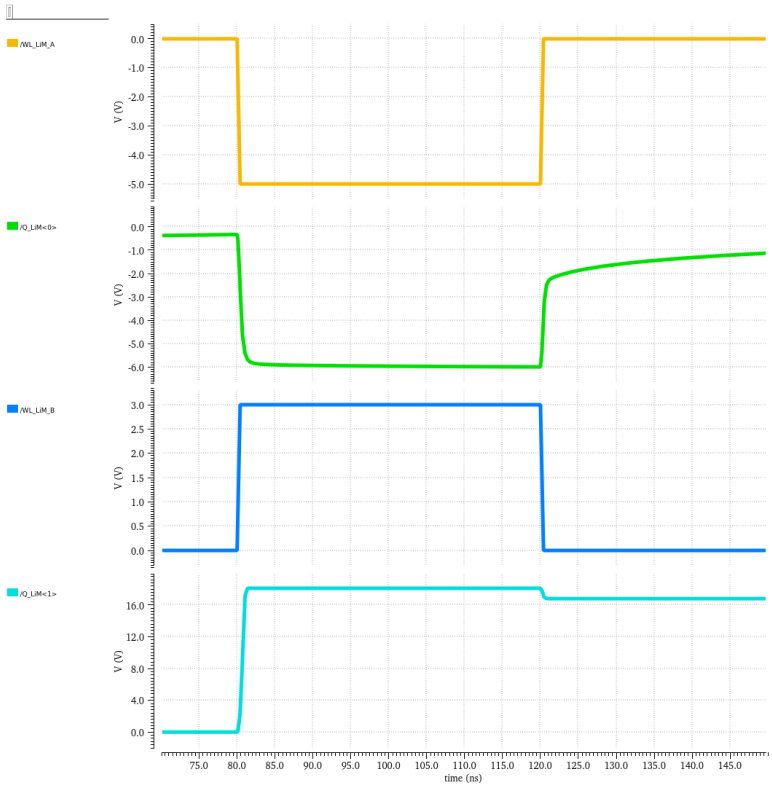


Figure 5.15: Setting up the LiM function by driving the two programming word lines. Simulation of the NOR function programmed to the 3–functions cell (1).

Simulation

The only simulation allowed is the one concerning the XOR function, and it is shown in Figure 5.22.

5.3.5 Full Adder cell

Schematic

The schematic of the LiM Full Adder cell is shown in Figure 5.23. This cell presents some differences with respect to the previous cells, starting from the fact that it takes three inputs and generates two outputs. Moreover, another difference is the presence of two logic subcirtcuits stacked one on the top of the other. That is, the `match line` of the first subcircuit is used as input for the second subcircuit.
In particular, the first logic subcircuit generates the signal $\overline{C_0}$ which is used

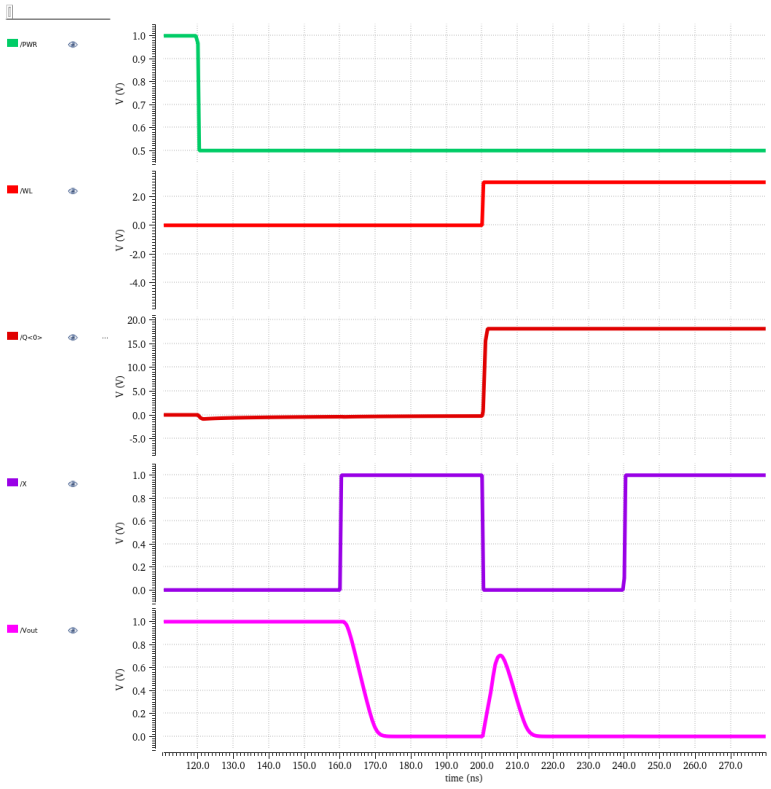


Figure 5.16: Executing the LiM function by changing `data` and `X`. Simulation of the NOR function programmed to the 3–functions cell (2).

as gate voltage to generate $\overline{S}$. Then, the outputs are finally converted by means of pull–up circuits and CMOS inverters, similarly to the other cells. Furthermore, this LiM cell is able to perform the full adder operation with its data as one addendum and two external data as the other addendum and the carry in.
The transistor number is 21 and this cell is the largest among the cells discussed in this work. Nonetheless, this full adder circuit can be also recognized as a programmable logic circuit if one of the inputs is fixed. That is, the two inputs AND, OR, XOR and XNOR functions are available in this LiM cell.

Simulation

Table 5.10 can be splitted in two parts, keeping one of the operands fixed. For instance:

- if $\mathtt{B} = 1$, `Sum` represents the XNOR function, while $\mathtt{C}_{\mathtt{out}}$ the OR function;

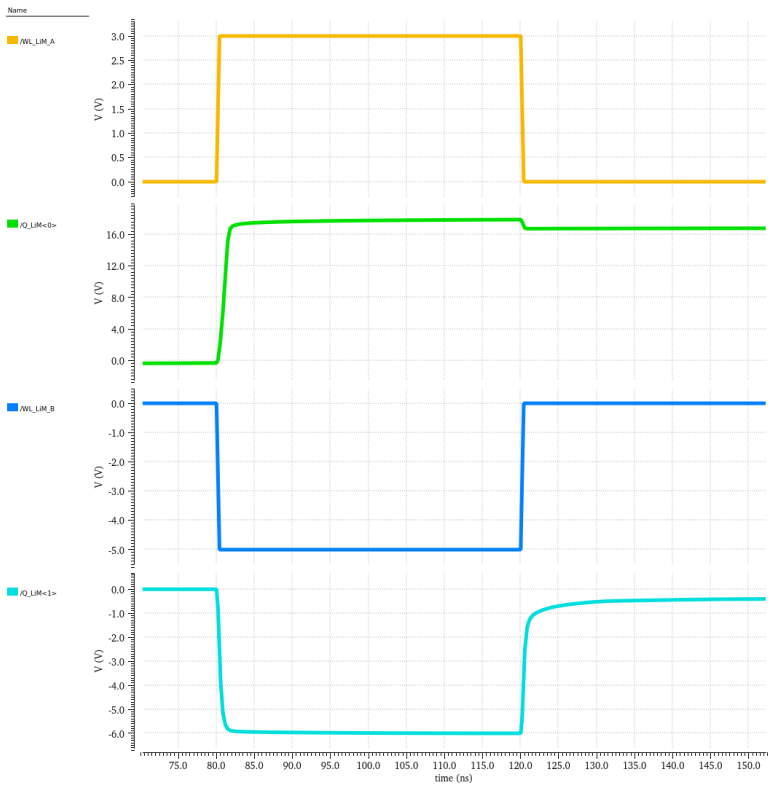


Figure 5.17: Setting up the LiM function by driving the two programming word lines. Simulation of the AND function programmed to the 3–funtions cell (1).

Word line(s)	single
External input number	2
Input line(s)	single
Logic functions	2 (4)
Transistor number	21

Table 5.9: Full Adder cell features.

- if `B` = 0, `Sum` represents the XOR function, while $\texttt{C}_{\texttt{out}}$ the AND function;

Therefore, these functions are simulated and shown in, respectively, Figure 5.24 and Figure 5.25.

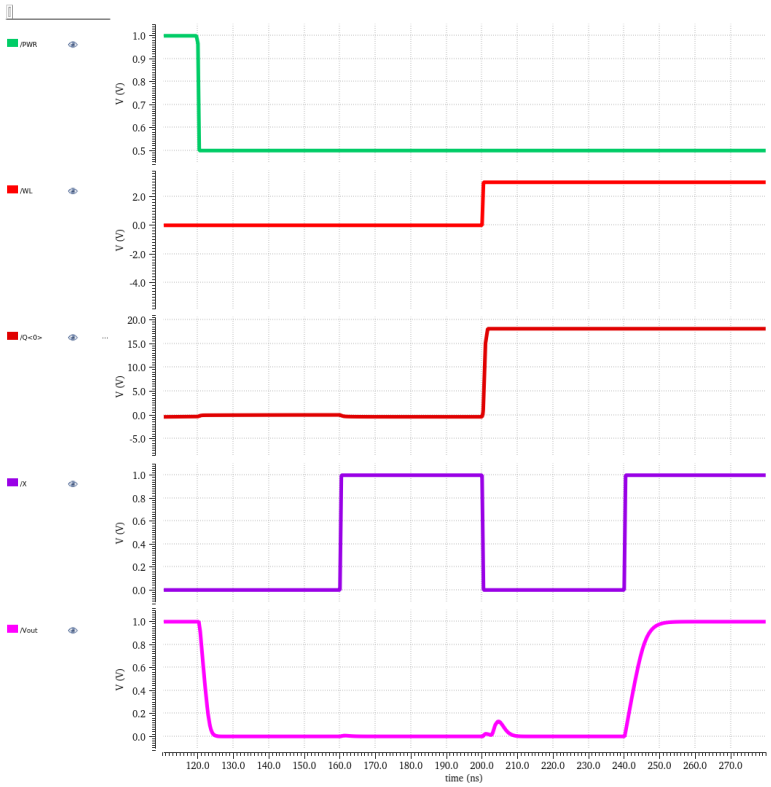


Figure 5.18: Executing the LiM function by changing `data` and `X`. Simulation of the AND function programmed to the 3–functions cell (2).

A	B	C_{in}	Sum	C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table 5.10: Truth table of the Full Adder cell.

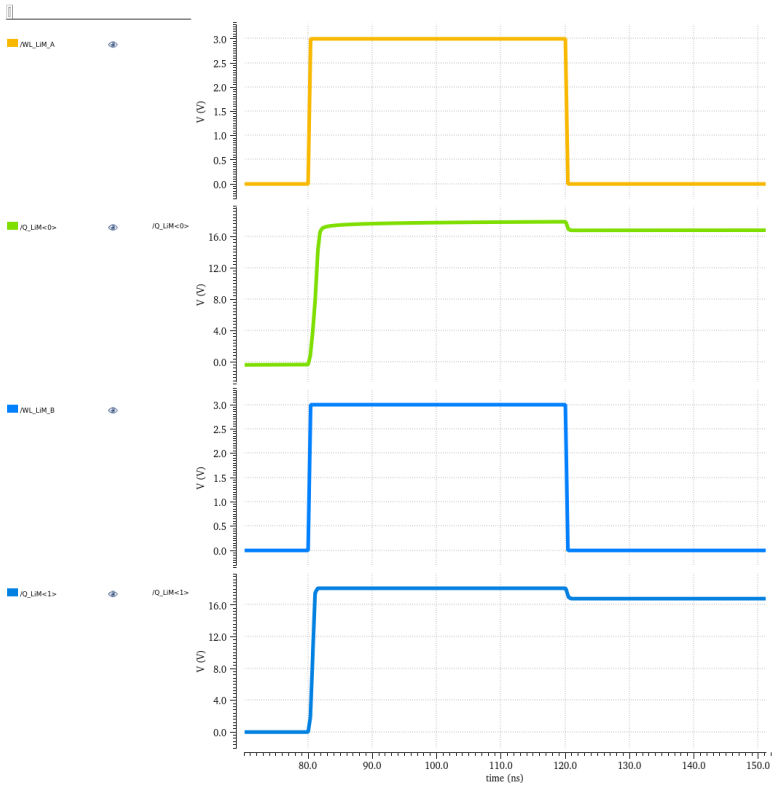


Figure 5.19: Setting up the LiM function by driving the two programming word lines. Simulation of the XNOR function programmed to the 3–functions cell (1).

5.3.6 Majority voter cell

Schematic

The concept behind the Majority Voter (MV) cell in Figure 5.26 is the same as in the case of the Full Adder cell. In fact, for the MV cell there is no possibility to program one ore more FeFETs to change logic function, but the presence of three total inputs — `data`, `B` and `C` — allows to have at disposal the two inputs functions AND and OR, in addition to the three inputs majority voter function.
Moreover, in this cell the logic circuit is made by three branches, since two sums and three products are needed to obtain the MV, and the number of transistor is 11.

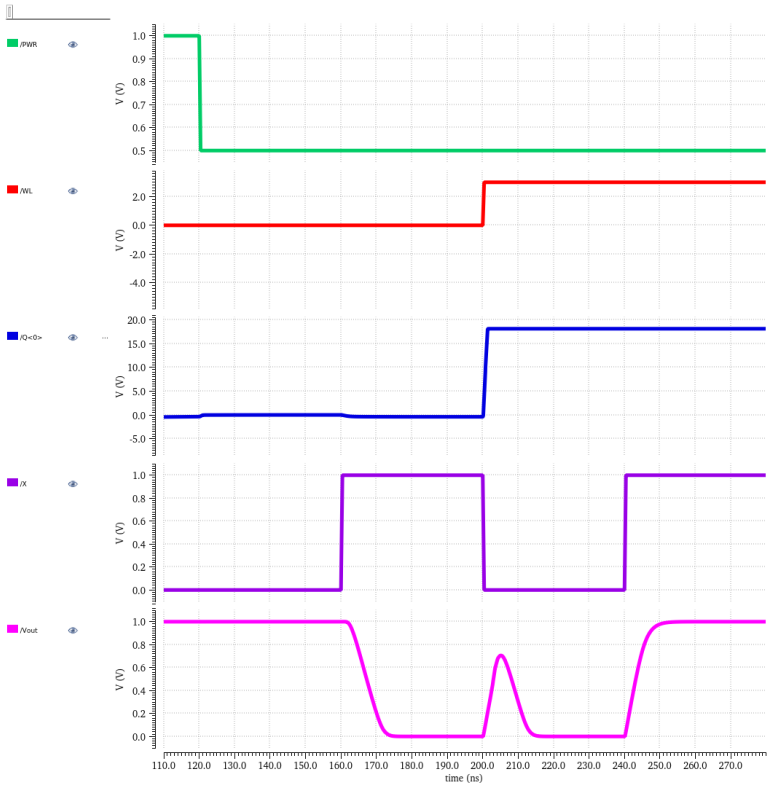


Figure 5.20: Executing the LiM function by changing `data` and `X`. Simulation of the XNOR function programmed to the 3–functions cell (2).

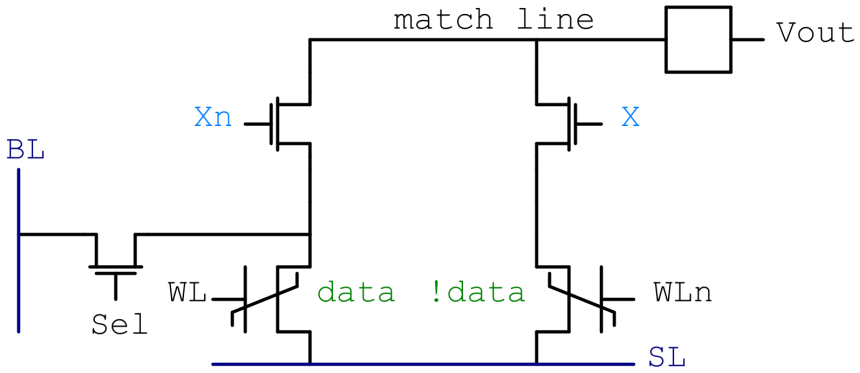


Figure 5.21: Schematic level of the LiM XOR cell.

Simulation

The same approach used for the Full Adder cell is used for the MV cell. That is, Table 5.12 is splitted in two parts and two functions are extracted:

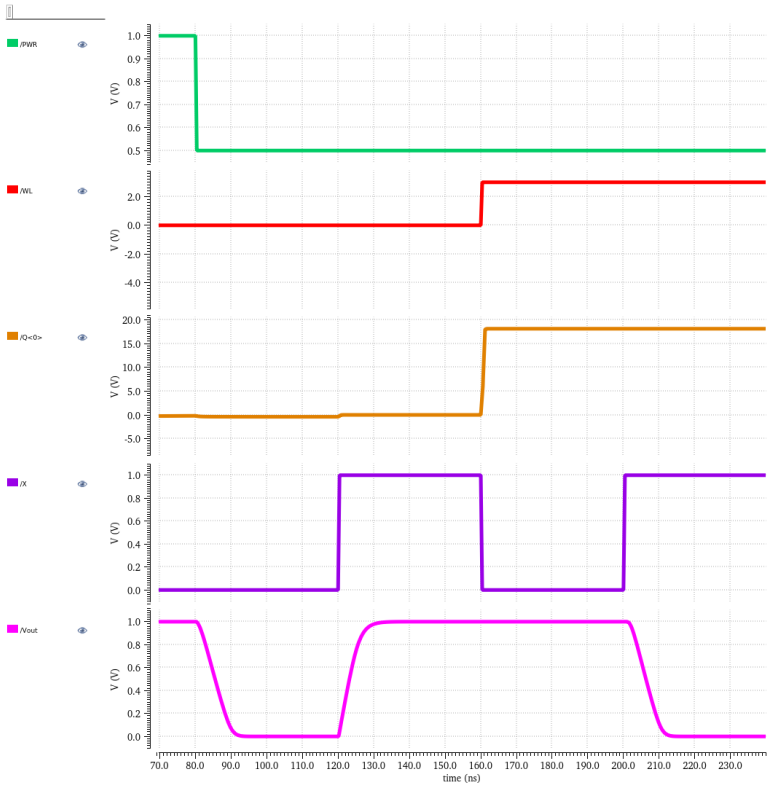


Figure 5.22: Executing the LiM function by changing `data` and `X`. Simulation of the XOR function in the XOR cell.

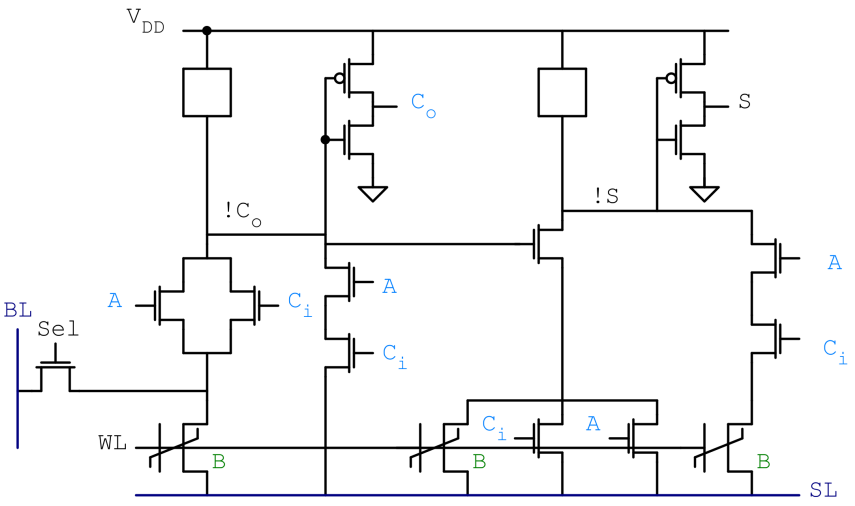


Figure 5.23: Schematic level of the LiM FA cell.

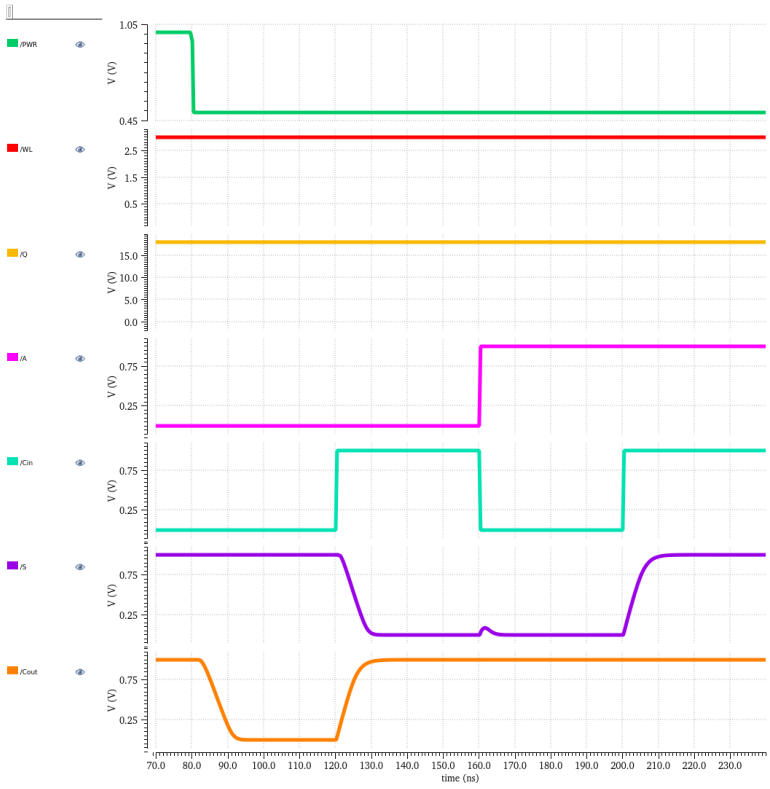


Figure 5.24: Executing the LiM function by changing A and C_{in}. B, which is the cell data, is fixed to '1'. Simulation of the XNOR and OR functions in the Full Adder cell.

Word line(s)	single
External input number	2
Input line(s)	single
Logic functions	1 (2)
Transistor number	11

Table 5.11: Majority Voter cell features.

- if A = 1, V_{out} represents the OR function;
- if A = 0, V_{out} represents the AND function.

Where A is the data stored in the cell and B and C are the external inputs. Therefore, these functions are simulated and shown in, respectively, Figure 5.27 and Figure 5.28. Finally, the cells features are compared in Table 5.13

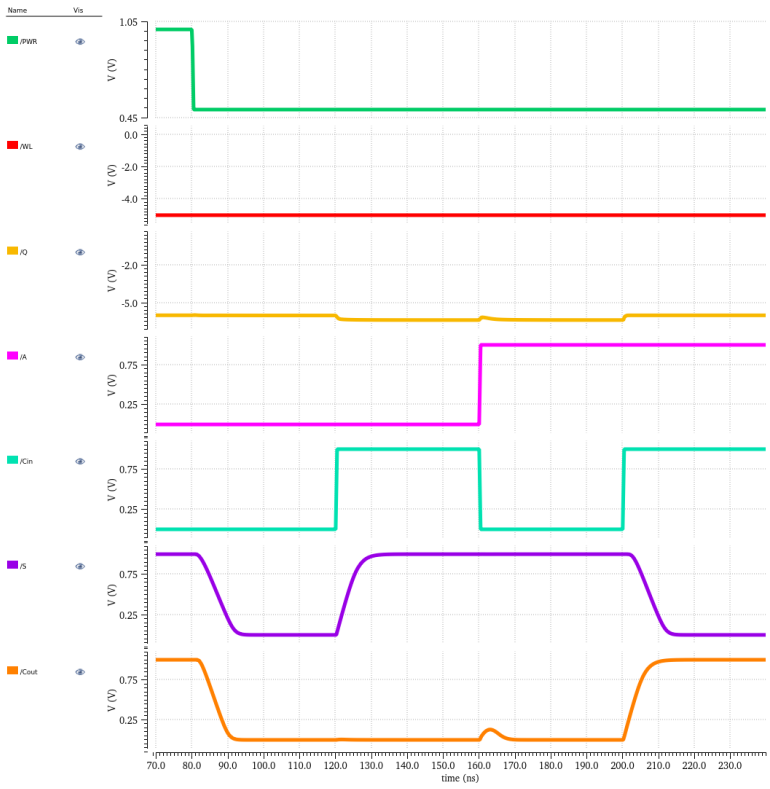


Figure 5.25: Executing the LiM function by changing `A` and $\texttt{C}_{\texttt{in}}$. `B`, which is the cell data, is fixed to '0'. Simulation of the XOR and AND functions in the Full Adder cell.

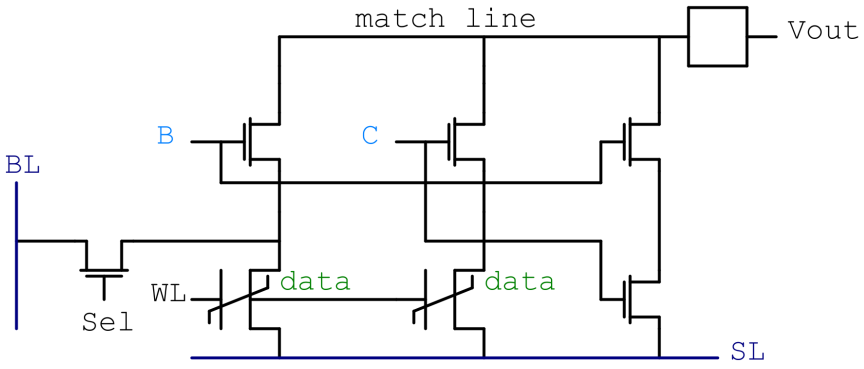


Figure 5.26: Schematic level of the LiM Majority Voter cell.

A	B	C	V_{out}
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Table 5.12: Truth table of the Majority Voter cell.

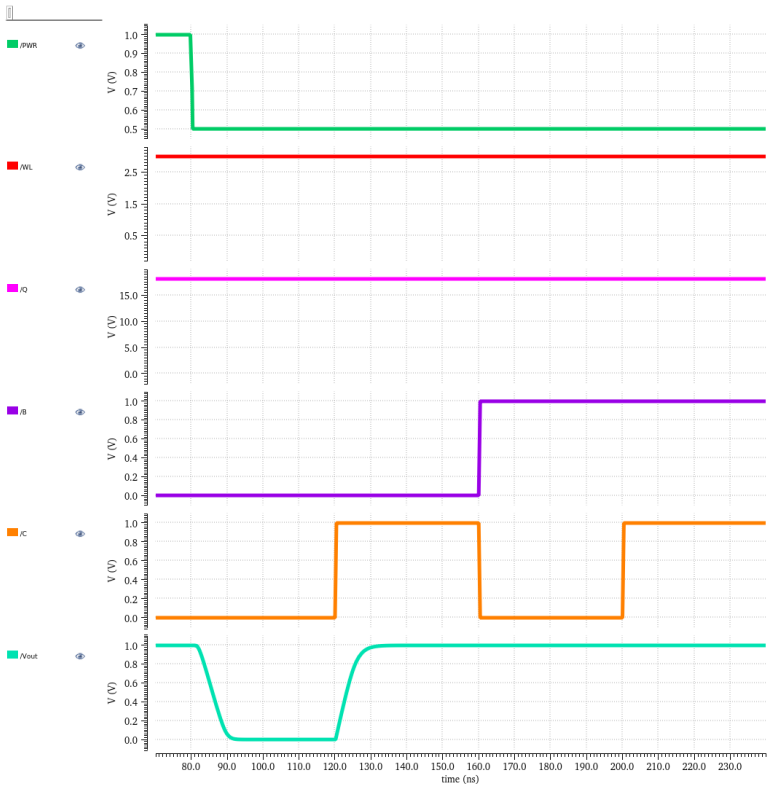


Figure 5.27: Executing the LiM function by changing B and C. A, which is the cell data, is fixed to '1'. Simulation of the OR function in the Majority voter cell.

5.4 Liberty characterization for Logic in Memory

5.4.1 Purposes 112

In the field of the simulation of logic gates, memory cells or ports in general, a library might usually be described in various formats, according to which

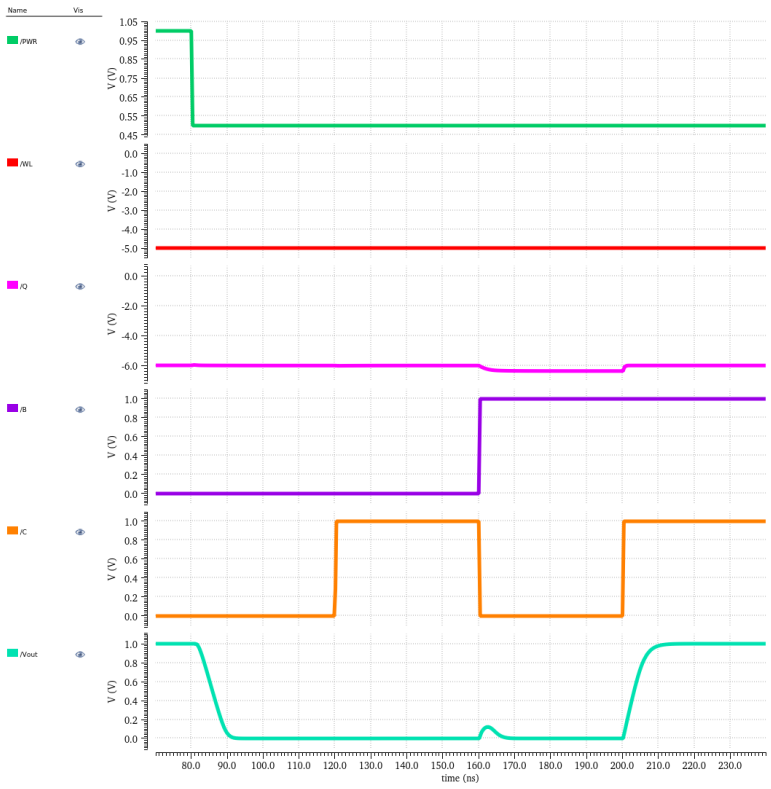


Figure 5.28: Executing the LiM function by changing `B` and `C`. `A`, which is the cell data, is fixed to '0'. Simulation of the AND function in the Majority voter cell.

tool is using it.
Therefore, the schematic level description, which is converted for instance by Cadence Virtuoso in a SPICE description, is mostly used in chapter 4 and section 5.3 for the purposes of this work. Nevertheless, commercial logic synthesis tools for digital microelectronics, in order to characterize an arbitrary architecture, use timing and power data to estimate:

- the best logic gate in terms of performance for a given path;
- the path total delay.

5.4.2 Liberty file template

As reported in [45], the aforementioned timing and power parameters are represented by a `.lib` file in ASCII language — Liberty file —, associated with a cell or gate in a precise semiconductor technology. In this case, the

	LUT	AND–OR	3–functions
Word line(s)	differential	single	differential
External input number	1	1	1
Input line(s)	differential	single	differential
Logic functions	14	2	3
Transistor number	15	11	11

(a)

	XOR	Full Adder	Majority Voter
Word line(s)	differential	single	single
External input number	1	2	2
Input line(s)	differential	single	single
Logic functions	1	2 (4)	1 (2)
Transistor number	9	21	11

(b)

Table 5.13: Comparison of cells dimension and features.

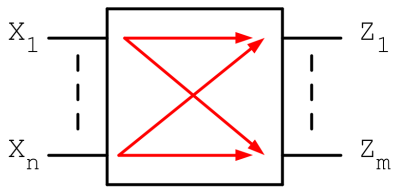


Figure 5.29: Concept of a cell in Liberty description. [45].

CMOS 28 nm FD–SOI.
Thereby, in order to obtain these performance descriptions, the cells have to be simulated under several input conditions, which is discussed in subsection 5.4.3.
Generally, a cell in this description is not different from a black box (Figure 5.29) characterized by its input and output signals and the paths between these. In particular, path delays are considered for each input signal transition which affects an output signal. Also, such a delay might depend on the state of the other inputs. Moreover, the LiM cells presented in section 5.3 do not have sequential logic, then each path does not depend on the other outputs.

```
/* General Syntax of a Technology Library */
library (nameoflibrary) {
... /* Library level simple and complex attributes */
... /* Library level group statements */
... /* Default attributes */
... /* Scaling Factors for delay calculation */

/* Cell definitions */

cell (cell_name) {
     ... /* cell level simple attributes */

     /* pin groups within the cell */
     pin(pin_name) {
      ... /* pin level simple attributes */

  /* timing group within the pin level */
  timing(){
    ... /* timing level simple attributes */
    } /* end of timing */

  ... /* aditional timing groups */

  } /* end of pin */

  ... /* more pin descriptions */

} /* end of cell */

... /* more cells */

} /* end of library */
```

Listing 5.1: Source code af a Liberty file template.

In Listing 5.1 it is shown a template of a Liberty file, where, first of all, the library name is defined. Then, cells are defined and for each pin the corrispondent time measurements are reported. That is, these are in the form of a look–up–table where delays information are provided, in relation with the simulation parameters. The same considerations are applied to the power measurements.

```
/* Units Attributes */
time_unit       :"1ns";
```

```
power_unit          :"1uW";
voltage_unit          :"1V";
current_unit          :"1A";
capacitive_load_unit       :"1pF";

/* Threshold Definitions */
slew_lower_threshold_pct_fall    :30.00 ;
slew_lower_threshold_pct_rise    :30.00 ;
slew_upper_threshold_pct_fall    :70.00 ;
slew_upper_threshold_pct_rise    :70.00 ;
input_threshold_pct_fall       :50.00 ;
input_threshold_pct_rise       :50.00 ;
output_threshold_pct_fall      :50.00 ;
output_threshold_pct_rise      :50.00 ;
```

Listing 5.2: Details of measurement units and thresholds of a Liberty file for the LiM cells in section 5.3.

Moreover, among the attributes of a library in a Liberty file, measurement units and thresholds are given in Listing 5.2. The discussed parameters are collected in Table 5.1.

```
pin(Vout) {
  direction    :output;
  timing() {
    related_pin    :"X";
    cell_rise(Timing_7_7) {
    index_1("0.05,0.1,0.15,0.2,0.25,0.3,0.35"); //input slew
   (ns)
    index_2 ("0.015,0.05,0.085,0.12,0.155,0.19,0.225"); //
   load capacitance (pF)
    values (.......);
    }
  }
}
```

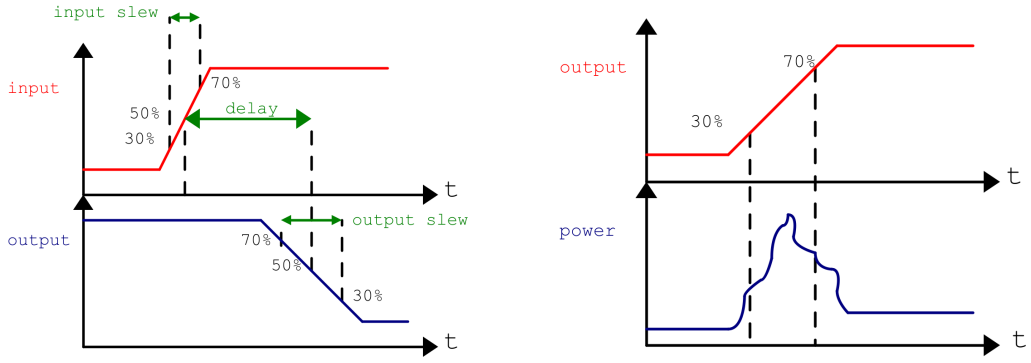
Listing 5.3: Example of look–up–table about timing measurements of a Liberty file for the LiM cells in section 5.3.

As far as the look–up–table is concerned, Listing 5.3 contains an example of how the timing measurements are organized. That is, in the section dedicated to the output pin V_{out}, a table for the output rise time is inserted in correspondence to the related pin (input) X. In particular, for each combination of the two simulation parameters — input slew and load capacitance — a timing value is given.

Therefore, this example applies to all the measurements which are discussed in subsection 5.4.3.

5.4.3 Measurements

For the objectives of this work, Liberate, the automatized tool in Cadence Virtuoso to perform Liberty simulations, cannot properly read the FeFET model discussed in subsection 3.1.1 since it is described in Verilog–A language.
Therefore, the simulator ADE available in Virtuoso has been adopted in order to obtain timing and power look–up–tables for each LiM cells in section 5.3.
Furthermore, each delay and power measurement follows the schemes in



(a) Input slew, output slew and delay.

(b) Dynamic power.

Figure 5.30: Liberty measurements in terms of delays and power [45].

Figure 5.30. In particular, the average dynamic power is calculated by integrating the power profile of the cell in the interval going from the 30% to the 70% of the rising (or falling) output. This interval is also calculated and collected as the *rise time* or *fall time* of the signal.
One must underline that every measurement is performed with parametric values of the input slew of the signals — 30%–70% rise or fall time — and of the load capacitance of the measured output. Therefore, these parameters are indicated in Listing 5.3 and are the same for every cell. Thus, 7x7 look–up–tables are generated.
Moreover, the list of measurements performed to the LiM cells is the following:

- **input to high output propagation delay**: every delay from the

rise/fall of one input to a rising output, when the former affects the latter;

- **input to low output propagation delay**: every delay from the rise/-fall of one input to a falling output, when the former affects the latter;
- **output rise time**: the 30%–70% rise time of an output;
- **output fall time**: the 70%–30% fall time of an output;
- **average dynamic power rising output**: the average dynamic power calculated during the output rise time;
- **average dynamic power falling output**: the average dynamic power calculated during the output fall time;
- **average static power**: the average static power calculated during one period;
- **peak power rising output**: the peak power when an output rises;
- **peak power falling output**: the peak power when an output falls;
- **word line to data '0' propagation delay**: the delay when '0' is written to a FeFET;
- **word line to data '1' propagation delay**: the delay when '1' is written to a FeFET;
- **writing data rise time**: the 30%–70% rise time of data;
- **writing data fall time**: the 70%–30% fall time of data;
- **average dynamic power programming**: the average dynamic power calculated during writing '1';
- **average dynamic power erasing**: the average dynamic power calculated during writing '0';

5.4.4 LiM cells

This section is a close–up to each LiM cell in section 5.3 in order to discuss the measurements to execute and to give examples of waveforms related to the Liberty characterization.

LUT cell

The LUT cell, recalling Figure 5.4, can be divided in 28 states, one for each logic functions for which the FeFETs can be programmed together with the data stored. Therefore, the variation of the input `X` is associated to the output V_{out}.
Moreover, Figure 5.31, Figure 5.32, Figure 5.33 and Figure 5.34 show waveforms from the Liberty simulation of the cell.

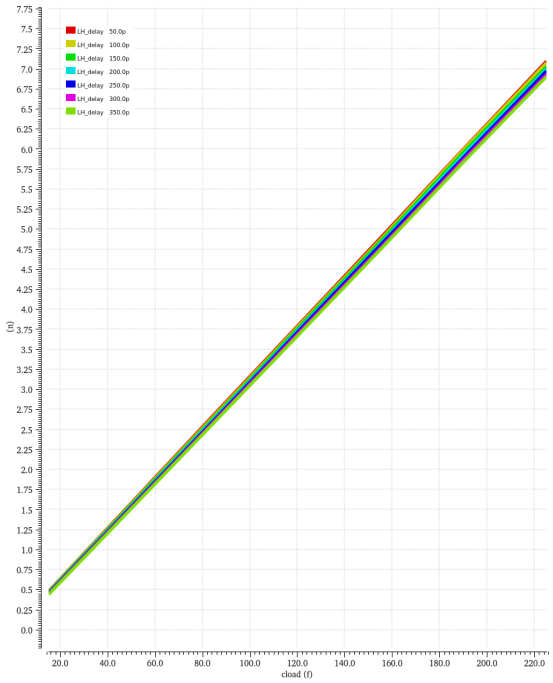


Figure 5.31: Low to high propagation delay (ns), state XOR, cell data '0' and `X` switching from '0' to '1'. LUT cell.

AND–OR cell

The AND–OR cell in Figure 5.9 can be divided in 8 states, one for each logic functions for which the FeFETs can be programmed together with the data stored. Therefore, the variation of the input `X` is associated to the output V_{out}.
Moreover, Figure 5.35 and Figure 5.36 show waveforms from the Liberty simulation of the cell.

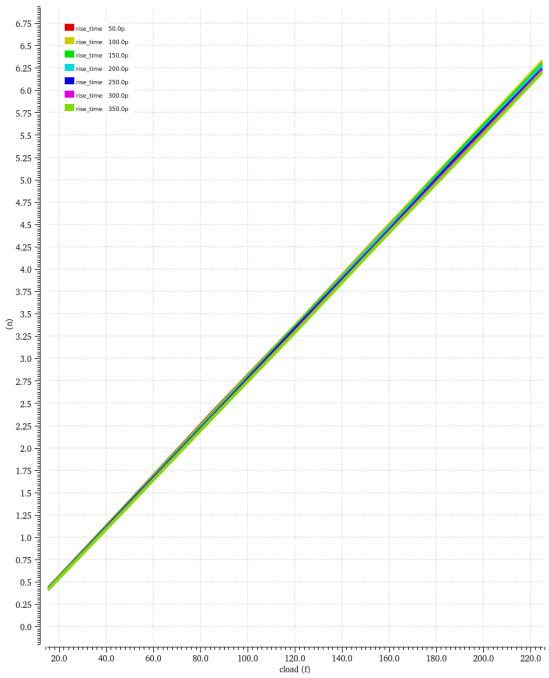


Figure 5.32: Output rise time (ns), state XOR, cell data '0' and `X` switching from '0' to '1'. LUT cell.

3–functions cell

The 3–functions cell, whose schematic is shown in Figure 5.14, can be divided in 6 states, one for each logic functions for which the FeFETs can be programmed together with the data stored. Therefore, the variation of the input `X` is associated to the output $\texttt{V}_{\texttt{out}}$.
Moreover, Figure 5.37, Figure 5.38 and Figure 5.39 show waveforms from the Liberty simulation of the cell.

XOR cell

The XOR cell, recalling Figure 5.21, can be divided in 2 states, one for each value of the data stored. Therefore, the variation of the input `X` is associated to the output $\texttt{V}_{\texttt{out}}$.
Moreover, Figure 5.40 and Figure 5.41 show waveforms from the Liberty simulation of the cell.

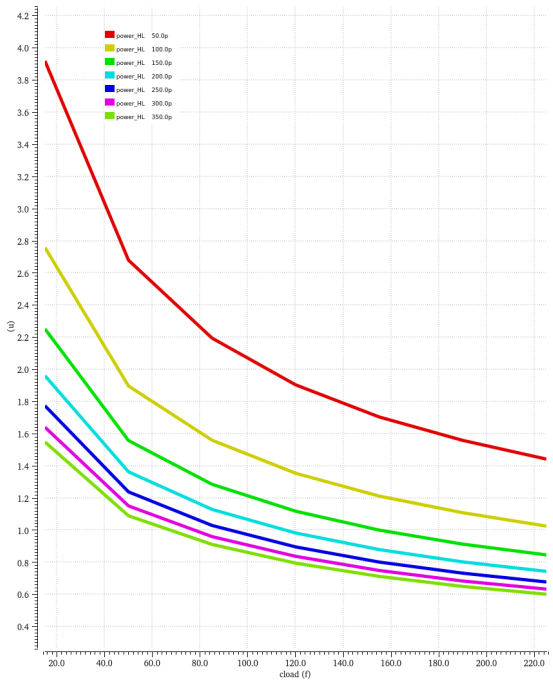


Figure 5.33: Average dynamic power (µW), state XOR, cell data '0' and `X` switching from '0' to '1'. LUT cell.

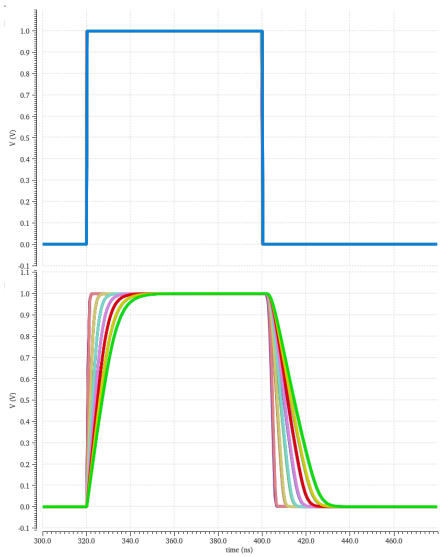


Figure 5.34: LiM output, state XOR, cell data '0' and `X` switching from '0' to '1'. LUT cell.

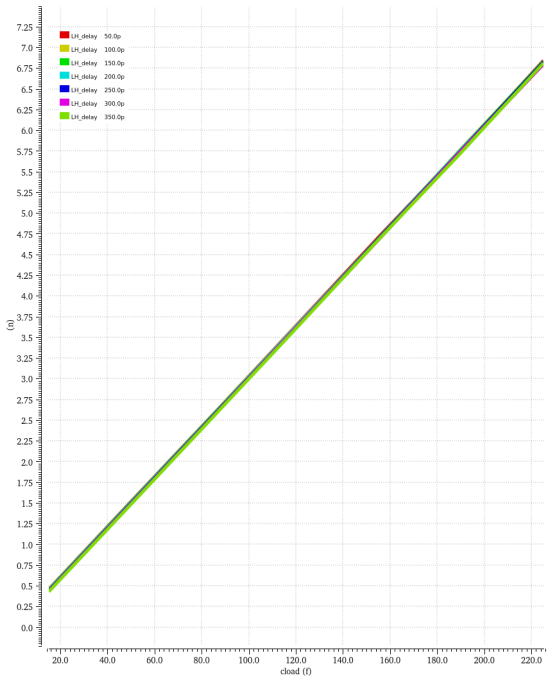


Figure 5.35: Low to high propagation delay (ns), state AND, cell data '1' and `X` switching from '0' to '1'. AND–OR cell.

Full Adder cell

The Full Adder cell, recalling Figure 5.23, can be divided in 2 states, one for each value of the data stored. Nevertheless, due to the presence of two external inputs, each measurement has to take into account that the other input might affect the outputs. Thus, 8 states for each output signal can be individuated. Then, the variation of the two inputs `A` and $\texttt{C}_{\texttt{in}}$ is associated to the outputs `Sum` and $\texttt{C}_{\texttt{out}}$.
Moreover, Figure 5.42 and Figure 5.43 show waveforms from the Liberty simulation of the cell.

Majority voter cell

The Majority Voter cell, shown in Figure 5.26, can be divided in 2 states, one for each value of the data stored. Nevertheless, due to the presence of two external inputs, each measurement have to take into account that the other input might affect the outputs. Thus, 8 states can be individuated. Then, the variation of the two inputs `B` and `C` is associated to the output $\texttt{V}_{\texttt{out}}$.
Moreover, Figure 5.44 and Figure 5.45 show waveforms from the Liberty

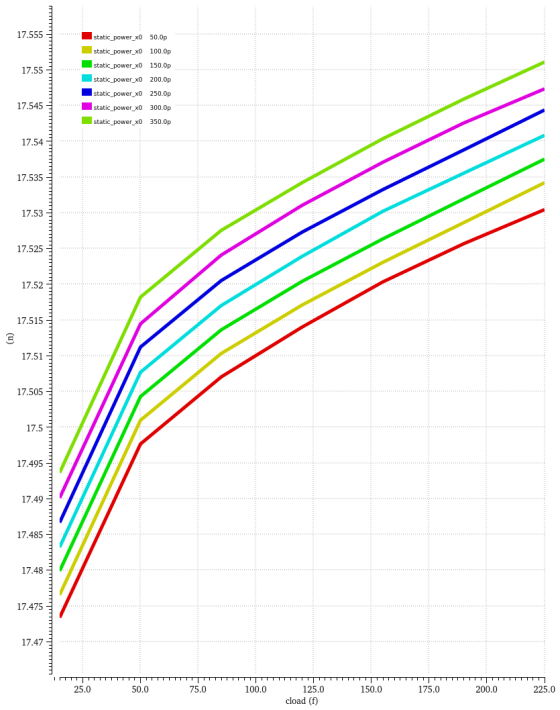


Figure 5.36: Average static power (nW), state AND, cell data '1' and V_{out} low. AND–OR cell.

simulation of the cell.

5.5 Results

At the end of the Liberty characterization performed in section 5.4, some consideration can be made by considering the extracted data. Therefore, for each measurement and each cell, a sample of data is randomly selected among one of the input–output combinations and then the worst value from all the available data is considered.

For what concerns the correlation number of transistors – number of functions (Figure 5.46), the LUT cell is clearly the most performant, although it provides only one output. Therefore, it is the most versatile. The Full Adder cell, instead, has the largest number of transistor but guarantees two outputs and an above the average number of functions.
Regarding the timing performance (Figure 5.47), the falling propagation delay appears to be always greater than the rising propagation delay. This

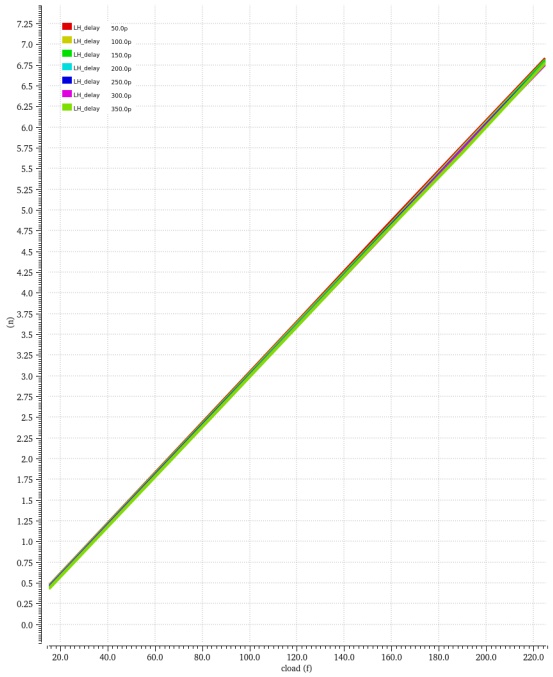


Figure 5.37: Low to high propagation delay (ns), state AND, cell data '1' and `X` switching from '0' to '1'. 3–functions cell.

aspect might be due to the fact that the output circuit in each cell is composed by a pull–up active load, as discussed in section 5.2. Thus, the output is driven to V_L with a greater effort.
Furthermore, a correlation exists between the peak power during a positive and a negative transition (Figure 5.48) and the distribution of the number of transistor. That is, the more transistors contribute to the logic function, the more instantaneus power is generated during the transitions.
As far as the average dynamic power during a falling transition is regarded, the power dissipated by the Full Adder cell and the Majority Voter cell is larger than the power dissipated by the other cells. This could be associated to the greater number of inputs and outputs with respect to the other cells.

5.6 Conclusions and future work

Logic in Memory cells have been realized exploiting the electrical properties of the ferroelectric field effect transistor, in the field of technologies beyond

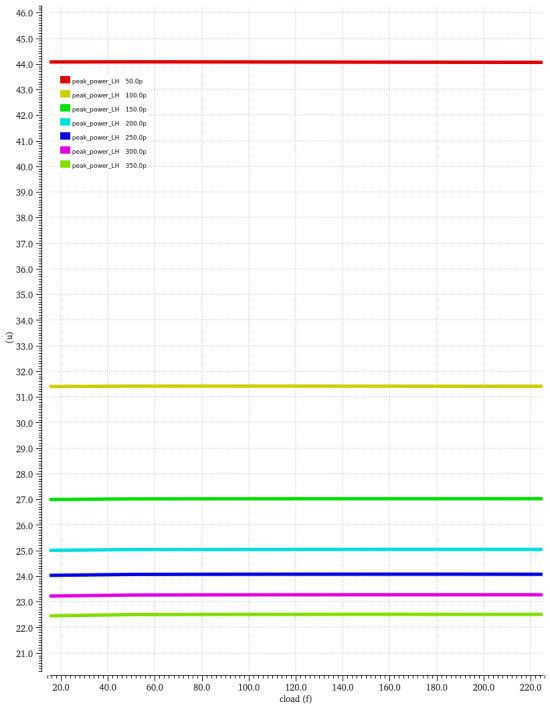


Figure 5.38: Low to high transition peak power (µW), state AND, cell data '1' and `X` switching from '0' to '1'. 3–functions cell.

the conventional CMOS. Six different cells have been characterized in Cadence Virtuoso in order to extract timing and power parameters for a Liberty file. Therefore, the concpet of Liberty characterization has been introduced in section 5.4.
The aforementioned cells have been presented at the schematic level and simulated. Results are shown in section 5.3.
Nevertheless, conventional memory arrays based on the ferroelectric transistor have been designed to propose an alternative to standard CMOS memories. These have been realized in different dimensions to discuss the effects of the scaling. Also, analog pheriperal circuits have been made. Results are discussed in subsection 4.5.1 and subsection 4.5.2.
Moreover, considerations regarding the LiM cells have been made in section 5.5.
Finally, this work could inspire future works and improvements, such as:

- in order to characterize the LiM cells in a more formal and automatized way, efforts might be made in the direction of converting the Verilog–A model of the FeFET into a SPICE description. Thus, Liberate tool

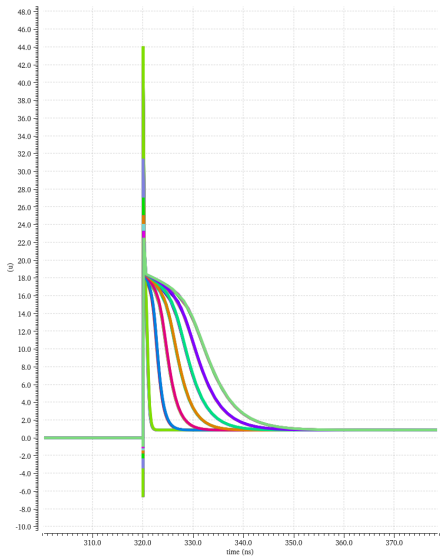


Figure 5.39: Power profile (µW), state AND, cell data ‘1’ and `X` switching from ‘0’ to ‘1’. 3–functions cell.

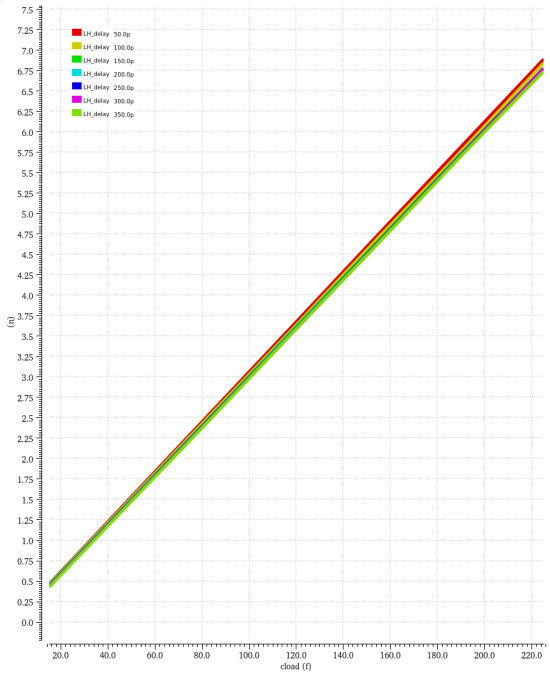


Figure 5.40: Low to high propagation delay (ns), cell data ‘0’ and `X` switching from ‘0’ to ‘1’. XOR cell.

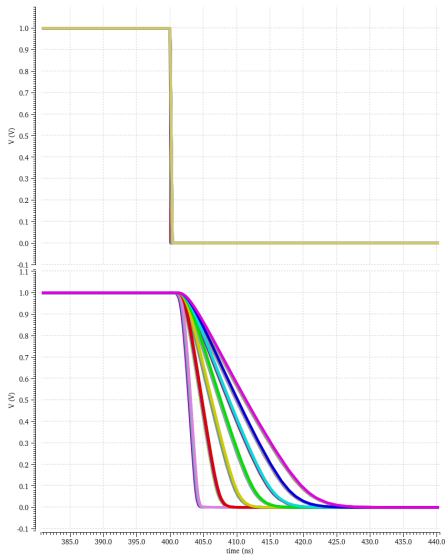


Figure 5.41: LiM output, cell data ‘0’ and `X` switching from ‘0’ to ‘1’. LUT cell. XOR cell.

would be able to read it and to generate the Liberty file which can be read by a synthesis tool;

- more LiM cells could be designed in order to expand the space of solutions for Logic in Memory which this work already offers;
- the use of a synthesis tool to generate a complete memory array starting from a LiM cell. Thus, a precise comparison between the performance of this LiM array and the conventional array discussed in chapter 4 becomes possible;
- a layout description of the FeFET, in first place, and of the LiM cells, secondly, is out of the goals of this work, but necessary to obtain more accurate measurements for the Liberty characterization;
- a more accurate physical model of the FeFET would provide a more realistic behavior of the device in terms of simulations. That is, the Preisach model mentioned in subsection 3.1.2 could be translated to a Verilog–A or SPICE model.

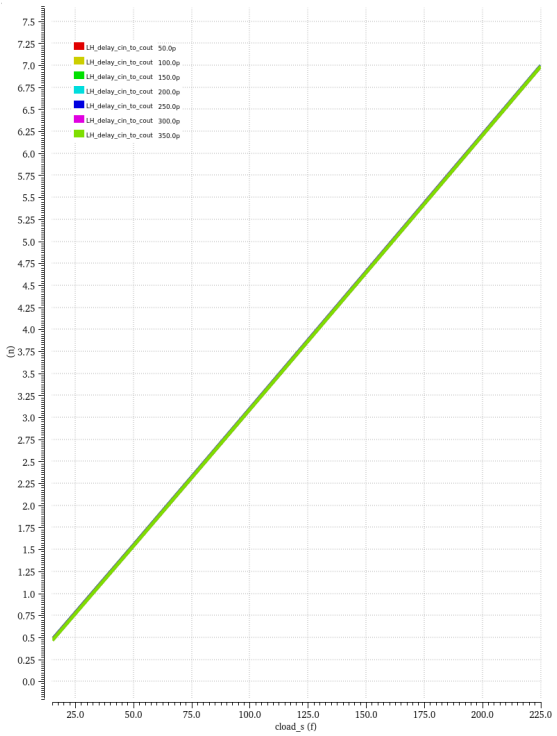


Figure 5.42: Low to high propagation delay (ns), cell data '0', $A = 1$ and $\mathtt{C_{in}}$ switching from '0' to '1'. Full Adder cell.

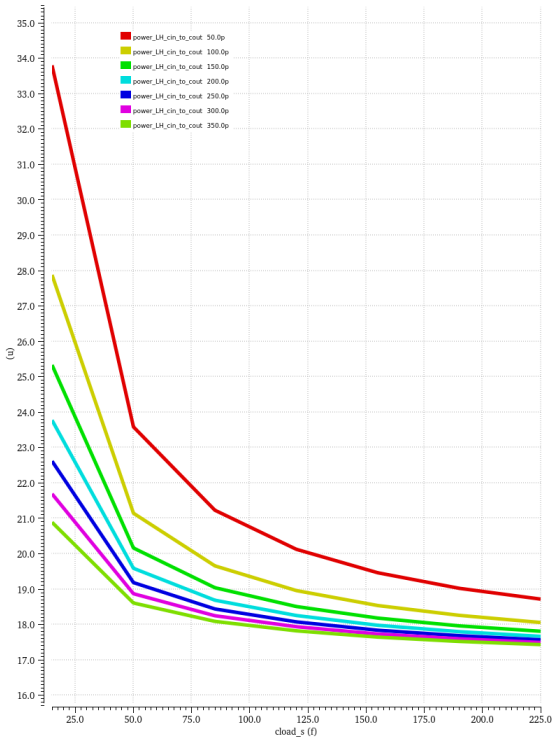


Figure 5.43: Average dynamic power (µW), cell data '0', $A = 1$ and $\mathtt{C_{in}}$ switching from '0' to '1'. Full Adder cell.

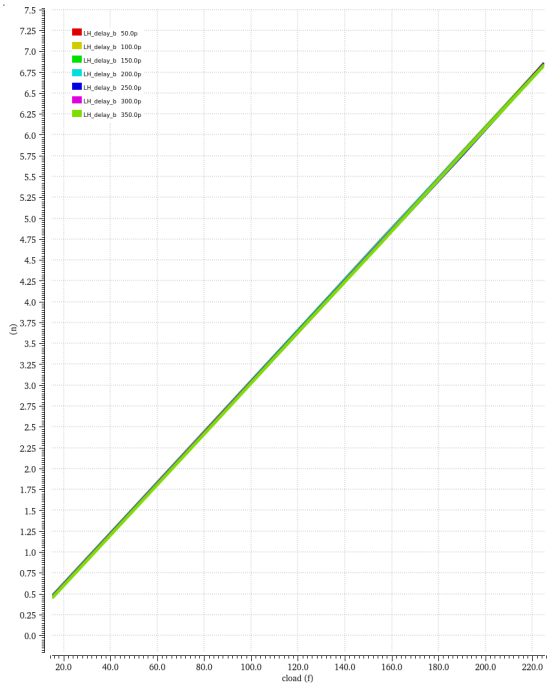


Figure 5.44: Low to high propagation delay (ns), cell data '0', $C = 1$ and `B` switching from '0' to '1'. Majority Voter cell.

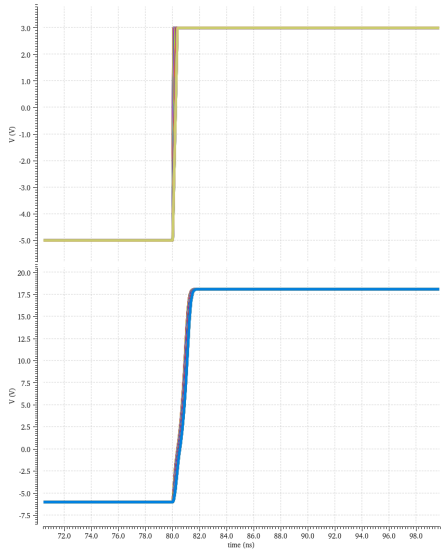


Figure 5.45: Cell data profile switching from '0' to '1'. Majority Voter cell.

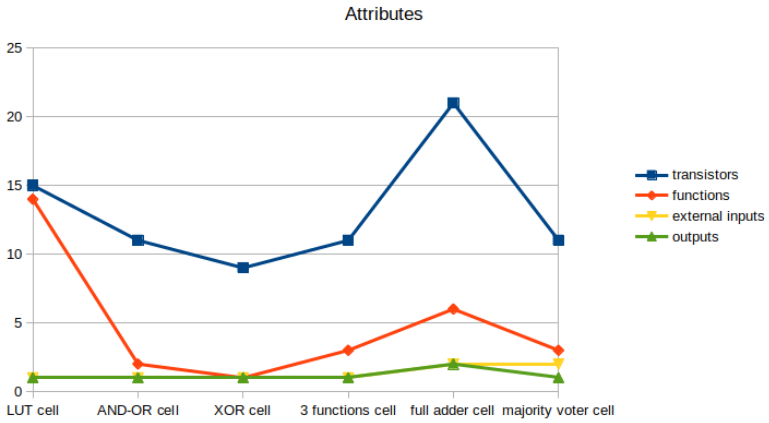


Figure 5.46: Prospect of cell size, number of functions, number of inputs and number of outputs from the different LiM cells.

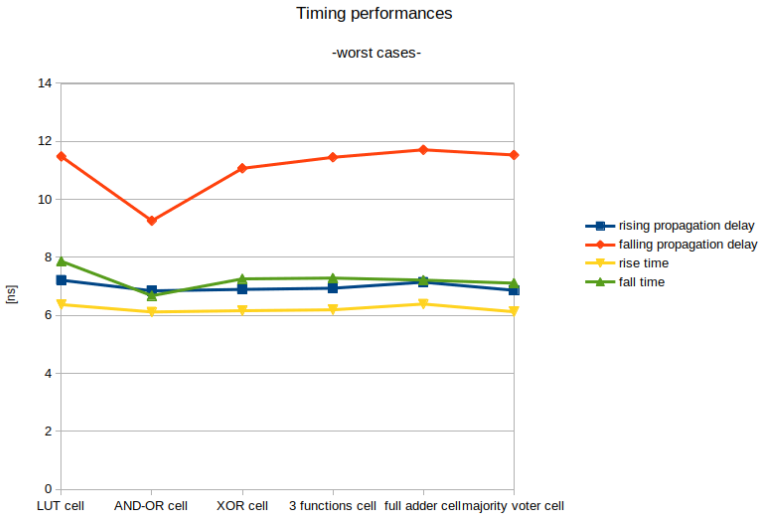


Figure 5.47: Prospect of the timing measurements from the different LiM cells.

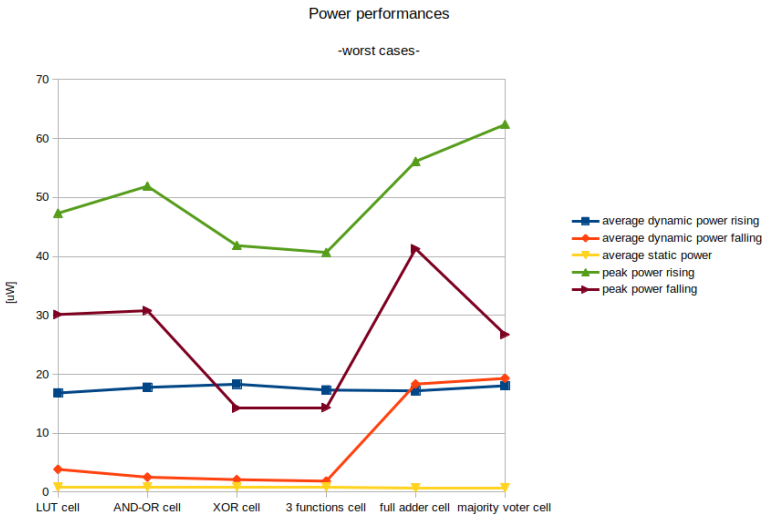


Figure 5.48: Prospect of the power measurements from the different LiM cells.

Appendix A

FeFET model Verilog–A source code

A.1 Ferroelectric capacitor

```
1  ////////////////////////////////////
2  //Copyright @ 2015 Purdue University
3
4  //The terms under which the software and associated
       documentation (the Software) is provided are as the
       following:
5
6  //The Software is provided "as is", without warranty of any
       kind, express or implied, including but not limited to the
        warranties of merchantability,
7  // fitness for a particular purpose and noninfringement. In
       no event shall the authors or copyright holders be liable
       for any claim, damages or other
8  // liability, whether in an action of contract, tort or
       otherwise, arising from, out of or in connection with the
       Software or the use or other dealings
9  //  in the Software.
10
11 //Purdue grants, free of charge, to any users the right to
       modify, copy, and redistribute the Software, both within
       the user's organization and
12 // externally, subject to the following restrictions:
13
14 //1. The users agree not to charge for the code itself but
       may charge for additions, extensions, or support.
15
```

```
16 //2. In any product based on the Software, the users agree to
       acknowledge the Negative capacitor model Research Group
   that developed the software. This
17 // acknowledgment shall appear in the product documentation.
18
19 //3. The users agree to obey all U.S. Government restrictions
       governing redistribution or export of the software.
20
21 //4. The users agree to reproduce any copyright notice which
   appears on the software on any copy or modification of
   such made available to others.
22
23 //Agreed to by
24 //Muhammad A. Wahab and Muhammad Ashraf Alam, Purdue
   University
25 //May 10, 2015
26 ////////////////////////////////////////////////////
27
28 /*****************************************************/
29 //revision log
30 /*****************************************************/
31 // Deployed on 04/05/2016 as a part of verilog-a NCFET model
   version 1.1.0 by Muhammad A. Wahab
32 // Added dipole response
33 /*****************************************************/
34 /*****************************************************/
35 // Deployed on 11/29/2015 as a part of verilog-a NCFET model
   version 1.0.0 by Muhammad A. Wahab
36 /*****************************************************/
37
38 ////////////////////////////////////////////////////
39 // Verilog-A version of Potential Model for negative
   capacitor
40 // Implemented on May 10, 2015 by Muhammad A. Wahab
41 // Default parameters are from References of the manual
42
43 `include "constants.vams"
44 `include "disciplines.vams"
45
46 module neg_cap_3t(ncp,ncn,qg_as_v);
47 inout ncp, ncn, qg_as_v;
48 electrical ncp, ncn, qg_as_v;
49
50
51 parameter real    alpha           = -1.05e11      from (-inf:
   inf);
```

```
52 parameter real     beta              = 1e17       from (-inf:inf)
   ;
53 parameter real     gamma             = 6e29         from (-inf:
   inf);
54 parameter real     rho            = 25           from [0:inf);
55 parameter real     tFE            = 100e-7       from [0:inf);
56 parameter real           W                 = 1e-4
                     from (0:inf);
57 parameter real     L                 = 45e-7
   from (0:inf);
58    parameter real    eps0 = 8.8542e-10 from (0:inf);
59
60    real      C0;
61    real      A;
62
63
64    branch (ncp, ncn) fecap, c0;
65
66
67 analog begin
68    A = W*L;
69    C0 = eps0*A/tFE;
70
71
72        //Potential across the ferroelectric capacitor, eq (5)
    of the manual
73        //1e-6 is used for unit conversion of charge: uCoul to
    Coul
74
75 V(fecap)    <+alpha*tFE*(V(qg_as_v)*1e-6) + beta*tFE*pow((V(
   qg_as_v)*1e-6),3.0) + gamma*tFE*pow((V(qg_as_v)*1e-6),5.0)
   ;
76    V(fecap)    <+ rho*tFE*ddt(V(qg_as_v)*1e-6);
77    I(c0) <+ C0*ddt(V(c0));
78
79        end
80 endmodule
```

Listing A.1: Ferroelectric capacitor in Verilog–A description. Reference to [29].

A.2 MOSFET

```
1 ////////////////////////////////////////////////
2 //Copyright @ 2015 Purdue University
3
```

```
4  //The terms under which the software and associated
       documentation (the Software) is provided are as the
       following:
5
6  //The Software is provided "as is", without warranty of any
       kind, express or implied, including but not limited to the
        warranties of merchantability,
7  // fitness for a particular purpose and noninfringement. In
       no event shall the authors or copyright holders be liable
       for any claim, damages or other
8  // liability, whether in an action of contract, tort or
       otherwise, arising from, out of or in connection with the
       Software or the use or other dealings
9  //  in the Software.
10
11 //Purdue grants, free of charge, to any users the right to
       modify, copy, and redistribute the Software, both within
       the user's organization and
12 // externally, subject to the following restrictions:
13
14 //1. The users agree not to charge for the code itself but
       may charge for additions, extensions, or support.
15
16 //2. In any product based on the Software, the users agree to
        acknowledge the Negative capacitor model Research Group
       that developed the software. This
17 // acknowledgment shall appear in the product documentation.
18
19 //3. The users agree to obey all U.S. Government restrictions
        governing redistribution or export of the software.
20
21 //4. The users agree to reproduce any copyright notice which
       appears on the software on any copy or modification of
       such made available to others.
22
23 //Agreed to by
24 //Muhammad A. Wahab and Muhammad Ashraf Alam, Purdue
       University
25 //May 10, 2015
26 ////////////////////////////////////////////////////////
27
28 /*****************************************************/
29 //revision log
30 /*****************************************************/
31 // Deployed on 04/05/2016 as a part of verilog-a NCFET model
       version 1.1.0 by Muhammad A. Wahab
```

```
// Declared the variables
// Modified Id leakage expression so that Id=0 at Vds=0
// Added a selector to inlcude or exclude the Id leakage
/*****************************************************/
/*****************************************************/
// Deployed on 11/29/2015 as a part of verilog-a NCFET model version 1.0.0 by Muhammad A. Wahab
// Added a dummy node to output the gate charge
// Added Id leakage with Id-Vgs of the previous MVS model
/*****************************************************/

//////////////////////////////////////////////////////
/* This file is for education and personal use only and is subject to the copyright of orginal publisher */
// Modified on May 10, 2015 by Muhammad A. Wahab to use as a part of NCFET model

/*****************************************************/
/* Original version of the MVS_1_0_1 model (collected from NEEDS nanohub website) */
/*****************************************************/
// VerilogA for virtual-source (VS) based self-consistent transport/capacitance model for Si MOSFET
// transport model:  A. Khakifirooz, et al, p. 1674, T-ED 2009.
// charge model: L. Wei et al, p. 1263, T-ED 2012.
// Implemented on July 15, 2013 by S. Rakheja
// Modified on Sep. 19, 2013 by S. Rakheja
/*****************************************************/

`include "constants.vams"
`include "disciplines.vams"

module mvs_5t_mod(d, g, s, b, qg_as_v);
inout d, g, s, b, qg_as_v;
electrical d, g, s, b, qg_as_v;
electrical di, si;

// Original VS parameters
parameter real    version   = 1.01;
                      //      MVS model version = 1.0.1
parameter integer   type    = 1       from [-1 : 1] exclude 0;      //  type of transistor. nFET type=1; pFET type=-1
parameter real    W           = 1e-4      from (0:inf);
          //  Transistor width [cm]
```

```
68 parameter real     Lgdr     = 80e-7      from (0:inf);
                //  Physical gate length [cm]. // This is the
   designed gate length for litho printing.
69 parameter real     dLg      = 10.5e-7      from (0:inf);
                  //  Overlap length including both source and
   drain sides [cm]
70 parameter real     Cg           = 2.2e-6     from (0:inf);
                    //  Gate-to-channel areal capacitance at
   the virtual source [F/cm^2]
71 parameter real     etov     = 1.3e-3     from (0:inf);
                //  Equivalent thickness of dielectric at S/D-G
    overlap [cm]
72 parameter real     delta      = 0.10       from [0:inf);
                  //  Drain-induced-barrier-lowering (DIBL) [V/
   V]
73 parameter real     n0           = 1.5        from [0:inf);
                    //  Subthreshold swing factor [unit-less] {
   typically between 1.0 and 2.0}
74 parameter real     Rs0      = 100        from (0:inf);
                //  Access resistance on s-terminal [Ohms-
   micron]
75 parameter real     Rd0      = 100        from (0:inf);
                //  Access resistance on d-terminal [Ohms-
   micron]
76
                              //  Generally, Rs0 = Rd0 for
   symmetric source and drain
77 parameter real     Cif      = 1e-12      from [0:inf);
                //  Inner fringing S or D capacitance [F/cm]
78 parameter real     Cof      = 2e-13      from [0:inf);
                //  Outer fringing S or D capacitance [F/cm]
79 parameter real     vxo      = 0.765e7      from (0:inf);
                  //  Virtual source injection velocity [cm/s]
80 parameter real     mu           = 200        from (0:inf);
                    //  Low-field mobility [cm^2/V.s]
81 parameter real     beta     = 1.7        from (0:inf);
                //  Saturation factor. Typ. nFET=1.8, pFET=1.6
82 parameter real       Tjun         = 298              from [173:
   inf);                      //    Junction temperature [K]
83 parameter real     phib     = 1.2;
                      //  ~abs(2*phif)>0 [V]
84 parameter real     gamma      = 0.0        from [0:inf);
                  //  Body factor  [sqrt(V)]
85 parameter real     Vt0      = 0.486;
                      //  Strong inversion threshold voltage [V
   ]
```

```
86 parameter real    alpha     = 3.5;
                        //  Empirical parameter for threshold
   voltage shift between strong and weak inversion.
87 parameter real    mc          = 0.2       from [0.01 : 10];
                    //  Choose an appropriate value between
   0.01 to 10
88
                              //  For, values outside of this
   range,convergence or accuracy of results is not guaranteed
89 parameter integer CTM_select  = 1               from [1 : inf
   );                     //  If CTM_select = 1, then classic
   DD-NVSAT model is used
90
                              //  For CTM_select other than 1,
   blended DD-NVSAT and ballistic charge transport model is
   used
91 parameter real    CC        = 0                    from [0:inf
   );                        //  Fitting parameter to adjust Vg-
   dependent inner fringe capacitances(Not used in this
   version)
92 parameter real    nd        = 0                    from [0:inf
   );                        //  Punch-through factor [1/V]
93
94 parameter integer leak_select = 1                     from
   [0:1];                       // If leak_select = 1, off-
   state leakage current saturation (with respect to Vgs) is
   included
95
96 `define  SMALL_VALUE  (1e-10)
97 `define  LARGE_VALUE  (40)
98
99 real Rs, Rd, Vds, Vgs, Vgsraw, Vgd, Vgdraw, Vbs, Vdsi, Vgsi,
   Vgdi, Vbsi, dir;
100 real Leff, me, S, phit;
101 real n, nphit, aphit, Vtpcorr, eVgpre, FFpre, ab, Vcorr,
   Vgscorr, Vbscorr, Vt0bs, Vt0bs0, Vtp, Vtp0;
102 real eVg, FF, eVg0, FF0, Qref, eta, eta0;
103 real Qinv, Qinv_corr, vx0, Vdsats, Vdsat,Vdratio, Vdbeta,
   Vdbetabeta, Fsat, Id ;
104 real Vgt, psis, Vgta, Vdsatq, Fsatq, x, den;
105 real qsc, qdc, qi, kq, kq2, kq4, tol, qsb, qdb, qs, qd, Qs,
   Qd;
106 real Qb, etai, Qinvi, dQinv, dibl_corr;
107 real Qinvs, Qinvd, Qsov, Qdov, Vt0x, Vt0y, Fs_arg, Fs, Fd_arg
   , Fd, FFx, FFy, Qsif, Qdif, Qg, a, Cofs, Cofd;
108
```

```
109 real Vt0bs0_vgs0, eVg_vgs0, FF_vgs0, eta_vgs0, Qinv_corr_vgs0
    , Vdsat_vgs0, Vdratio_vgs0, Vdbeta_vgs0, Vdbetabeta_vgs0,
    Fsat_vgs0, Id_vgs0;
110
111 analog begin
112
113   //Voltage definitions
114   Vgsraw      = type * ( V(g) - V(si) );
115   Vgdraw      = type * ( V(g) - V(di) );
116   if (Vgsraw >= Vgdraw) begin
117     Vds    = type * ( V(d) - V(s) );
118     Vgs    = type * ( V(g) - V(s) );
119     Vbs    = type * ( V(b) - V(s) );
120     Vdsi     = type * ( V(di) - V(si) );
121     Vgsi     = Vgsraw;
122     Vbsi     = type * ( V(b) - V(si) );
123     dir    = 1;
124   end
125   else begin
126     Vds    = type * ( V(s) - V(d) );
127     Vgs    = type * ( V(g) - V(d) );
128     Vbs    = type * ( V(b) - V(d) );
129     Vdsi     = type * ( V(si) - V(di) );
130     Vgsi   = Vgdraw;
131     Vbsi   = type * ( V(b) - V(di) );
132     dir    = -1;
133   end
134
135
136     //Parasitic element definition
137   Rs       = 1e-4/ W * Rs0;
              //  s-terminal resistance [ohms]
138   Rd       = Rs;
              //  d-terminal resistance [ohms] For symmetric
    source and drain Rd = Rs.
139      //Rd       = 1e-4/ W * Rd0;
                  //  d-terminal resistance [ohms] {Uncomment
    for asymmetric source and drain resistance.}
140     Cofs       =    ( 0.345e-12/ etov ) * dLg/ 2.0 + Cof;
         //  s-terminal outer fringing cap [F/cm]
141     Cofd       =    ( 0.345e-12/ etov ) * dLg/ 2.0 + Cof;
         //  d-terminal outer fringing cap [F/cm]
142   Leff       =    Lgdr - dLg;
              //  Effective channel length [cm]. After
    subtracting overlap lengths on s and d side
143
```

```
144     phit              =     $vt(Tjun);
                                //  Thermal voltage, kT/q [V]
145   me          = (9.1e-31) * mc;
                    //  Carrier mass [Kg]
146   n           = n0 + nd * Vds;
                    //  Total subthreshold swing factor taking
     punchthrough into account [unit-less]
147   nphit       =   n * phit;
                    //  Product of n and phit [used as one variable
     ]
148   aphit       = alpha * phit;
                   //  Product of alpha and phit [used as one
     variable]
149
150
151   //Correct Vgsi and Vbsi
152   //Vcorr is computed using external Vbs and Vgs but internal
       Vdsi, Qinv and Qinv_corr are computed with uncorrected
     Vgs, Vbs and corrected Vgs, Vbs respectively.
153   Vtpcorr     = Vt0 + gamma * (sqrt(abs(phib - Vbs))- sqrt(
     phib))- Vdsi * delta;//  Calculated from extrinsic Vbs
154   eVgpre      =   exp(( Vgs - Vtpcorr )/ ( aphit * 1.5 ));
          //  Calculated from extrinsic Vgs
155     FFpre       =   1.0/ ( 1.0 + eVgpre );            //
     Only used to compute the correction factor
156     ab          = 2 * ( 1 - 0.99 * FFpre ) * phit;
157     Vcorr       = ( 1.0 + 2.0 * delta ) * ( ab/ 2.0 ) * ( exp(
     -Vdsi/ ab ));  //  Correction to intrinsic Vgs
158   Vgscorr     = Vgsi + Vcorr;               //  Intrinsic Vgs
       corrected (to be used for charge and current computation)
159   Vbscorr     = Vbsi + Vcorr;               //  Intrinsic Vgs
       corrected (to be used for charge and current computation)
160   Vt0bs       = Vt0 + gamma * (sqrt( abs( phib - Vbscorr)) -
     sqrt( phib ));   //  Computed from corrected intrinsic Vbs
161   Vt0bs0      = Vt0 + gamma * (sqrt( abs( phib - Vbsi)) -
     sqrt( phib ));  //  Computed from uncorrected intrinsic
     Vbs
162   Vtp         = Vt0bs - Vdsi * delta - 0.5 * aphit;
     //  Computed from corrected intrinsic Vbs and intrinsic
     Vds
163     Vtp0        = Vt0bs0 - Vdsi * delta - 0.5 * aphit;
     //  Computed from uncorrected intrinsic Vbs and intrinsic
     Vds
164   eVg         = exp(( Vgscorr - Vtp )/ ( aphit ));        //
       Compute eVg factor from corrected intrinsic Vgs
165   FF          = 1.0/ ( 1.0 + eVg );
```

```
166   eVg0       = exp(( Vgsi - Vtp0 )/ ( aphit ));         //
       Compute eVg factor from uncorrected intrinsic Vgs
167   FF0          = 1.0/ ( 1.0 + eVg0 );
168   Qref       = Cg * nphit;
169   eta          = ( Vgscorr - ( Vt0bs - Vdsi * delta - FF *
       aphit ))/ ( nphit );  //  Compute eta factor from
       corrected intrinsic Vgs and intrinsic Vds
170   eta0       = ( Vgsi - ( Vt0bs0 - Vdsi * delta - FFpre *
       aphit ))/ ( nphit ); //  Compute eta0 factor from
       uncorrected intrinsic Vgs and internal Vds.
171                          //  Using FF instead of FF0 in eta0
        gives smoother capacitances.
172
173
174     //Charge at VS in saturation (Qinv)
175       if (eta  <= `LARGE_VALUE) begin
176     Qinv_corr    =    Qref * ln( 1.0 + exp(eta) );
177   end
178   else begin
179     Qinv_corr = Qref * eta;
180   end
181   if (eta0 <= `LARGE_VALUE) begin
182     Qinv     = Qref * ln( 1.0 + exp(eta0) );          //
       Compute charge w/ uncorrected intrinsic Vgs for use later
       on in charge partitioning
183   end
184   else  begin
185     Qinv     = Qref * eta0;
186   end
187
188   //Transport equations
189   vx0      = vxo;
190   Vdsats       = vx0 * Leff/ mu;
191   Vdsat      = Vdsats * ( 1.0 - FF ) + phit * FF;         //
       Saturation drain voltage for current
192   Vdratio      = abs( Vdsi/ Vdsat);
193   Vdbeta       = pow( Vdratio, beta);
194   Vdbetabeta     =    pow( 1.0 + Vdbeta, 1.0/ beta);
195   Fsat       = Vdratio / Vdbetabeta;            //  Transition
       function from linear to saturation.
196                          //  Fsat = 1 when Vds>>Vdsat; Fsat=
        Vds when Vds<<Vdsat
197
198     //Total drain current
199     //******Thermionic current at zero gate bias (Vgs=0)
       *********//
```

```
200     //added by Muhammad A. Wahab of Purdue University on May
    10, 2015, revised on Mar 31, 2016
201     if (leak_select == 1) begin
202        Vt0bs0_vgs0      = Vt0 + gamma * (sqrt( abs( phib -
    Vbs)) - sqrt( phib ));
203        eVg_vgs0         = exp((0 - ( Vt0bs0_vgs0 - Vds *
    delta-0.5*aphit ) )/ ( aphit ));
204       FF_vgs0           = 1.0/ ( 1.0 + eVg_vgs0 );
205        eta_vgs0         = ( 0 - ( Vt0bs0_vgs0 - Vds * delta -
     aphit*FF_vgs0 ))/ ( nphit );
206        Qinv_corr_vgs0       =   Qref * ln( 1.0 + exp(eta_vgs0
    ) );
207
208        Vdsat_vgs0     = Vdsats * ( 1.0 - FF_vgs0 ) + phit *
    FF_vgs0;     //   Saturation drain voltage for current at
    Vgs=0
209       Vdratio_vgs0       = abs( Vds/ Vdsat_vgs0);
210       Vdbeta_vgs0      = pow( Vdratio_vgs0, beta);
211       Vdbetabeta_vgs0      =   pow( 1.0 + Vdbeta_vgs0, 1.0/
    beta);
212       Fsat_vgs0      = Vdratio_vgs0 / Vdbetabeta_vgs0;
          //   Transition function from linear to saturation at
     Vgs=0
213        Id_vgs0              =   Qinv_corr_vgs0 * vx0 *
    Fsat_vgs0 * W;
214
215     end
216     else  begin
217        Id_vgs0              =   0;
218     end
219
220
221     //******Total drain current*********//
222     Id               = Qinv_corr * vx0 * Fsat * W + Id_vgs0;
223
224   //Calculation of intrinsic charge partitioning factors (qs
     and qd)
225   Vgt      = Qinv/ Cg;               //  Use charge computed
     from uncorrected intrinsic Vgs
226
227   // Approximate solution for psis is weak inversion
228   if (gamma == 0) begin
229     a      =   1.0;
230     if (eta0 <= `LARGE_VALUE) begin
231                psis     =   phib + phit * ( 1.0 + ln( ln( 1.0
     + `SMALL_VALUE + exp( eta0 ))));
```

```
232     end
233     else begin
234                 psis    = phib + phit * ( 1.0 + ln( eta0 ));
235           end
236   end
237   else begin
238           if (eta0 <= `LARGE_VALUE) begin
239               psis  = phib + ( 1.0 - gamma )/ ( 1.0 + gamma )
      * phit * ( 1.0 + ln( ln( 1.0 + `SMALL_VALUE + exp( eta0 )
     )));
240           end
241           else begin
242                 psis  = phib + ( 1.0 - gamma )/ ( 1.0 + gamma
      ) * phit * ( 1.0 + ln( eta0 ));
243           end
244           a   = 1.0 + gamma/ ( 2.0 * sqrt( abs( psis - ( Vbsi
      ))));
245       end
246   Vgta      = Vgt/ a;                 //  Vdsat in strong
     inversion
247   Vdsatq      = sqrt( FF0 * aphit * aphit + Vgta * Vgta);
       //  Vdsat approx. to extend to weak inversion;
248                           //  The multiplier of phit has
     strong effect on Cgd discontinuity at Vd=0.
249
250   // Modified Fsat for calculation of charge partitioning
251   //DD-NVSAT charge
252   Fsatq     = abs( Vdsi/ Vdsatq )/ ( pow( 1.0 + pow( abs(
     Vdsi/ Vdsatq ), beta ), 1.0/ beta ));
253   x     = 1.0 - Fsatq;
254   den     = 15 * ( 1 + x ) * ( 1 + x );
255   qsc       = Qinv *(6 + 12 * x + 8 * x * x + 4 * x * x * x)/
       den;
256   qdc     = Qinv *(4 + 8 * x + 12 * x * x + 6 * x * x * x)/
     den;
257   qi      = qsc + qdc;              //  Charge in the channel
258
259
260   //QB charge
261   kq      =   0.0;
262   tol     =   ( `SMALL_VALUE * vxo/ 100.0 ) * ( `SMALL_VALUE
     * vxo/ 100.0 ) * me/ ( 2 * `P_Q );
263   if (Vdsi <= tol) begin
264     kq2   = ( 2.0 * `P_Q/ me * Vdsi )/ ( vx0 * vx0 ) *
     10000.0;
265     kq4   = kq2 * kq2;
```

```
266     qsb    = Qinv * ( 0.5 - kq2/ 24.0 + kq4/ 80.0 );
267     qdb    = Qinv * ( 0.5 - 0.125 * kq2 + kq4/ 16.0 );
268   end
269   else begin
270     kq     = sqrt( 2.0 * `P_Q/ me * Vdsi )/ vx0 * 100.0;
271     kq2    = kq * kq;
272     qsb    = Qinv * ( asinh( kq )/ kq - ( sqrt( kq2 + 1.0 ) -
      1.0 )/ kq2);
273     qdb    = Qinv * (( sqrt( kq2 + 1.0 )- 1.0 )/ kq2);
274   end
275
276
277   // Flag for classic or ballistic charge partitioning:
278   if (CTM_select == 1) begin                  //  Ballistic
      blended with classic DD-NVSAT
279     qs     = qsc;                 //  Calculation of "ballistic
      " channel charge partitioning factors, qsb and qdb.
280     qd     = qdc;                 //  Here it is assumed that
      the potential increases parabolically from the
281   end                         //  virtual source point, where
      Qinv_corr is known to Vds-dvd at the drain.
282   else begin                          //  Hence carrier velocity
      increases linearly by kq (below) depending on the
283     qs     = qsc * ( 1 - Fsatq * Fsatq ) + qsb * Fsatq * Fsatq
      ;    //  efecive ballistic mass of the carriers.
284     qd     = qdc * ( 1 - Fsatq * Fsatq ) + qdb * Fsatq * Fsatq
      ;
285   end
286
287
288   //Body charge based on approximate surface potential (psis)
       calculation with delta=0 using psis=phib in Qb gives
      continuous Cgs, Cgd, Cdd in SI, while Cdd is smooth anyway
      .
289   Qb       = -type * W * Leff * ( Cg * gamma * sqrt( abs( psis
       - Vbsi )) + ( a - 1.0 )/ ( 1.0 * a ) * Qinv * ( 1.0 - qi
      ));
290
291   //DIBL effect on drain charge calculation.
292   //Calculate dQinv at virtual source due to DIBL only. Then:
      Correct the qd factor to reflect this channel charge
      change due to Vd
293   //Vt0bs0 and FF=FF0 causes least discontinuity in Cgs and
      Cgd but produces a spike in Cdd at Vds=0 (in weak
      inversion.  But bad in strong inversion)
294   etai       = ( Vgsi - ( Vt0bs0 - FF * aphit ))/ ( nphit );
```

```
295   if (etai <= `LARGE_VALUE) begin
296     Qinvi    = Qref * ln( 1.0 + exp( etai ));
297   end
298   else begin
299     Qinvi    = Qref * etai;
300   end
301   dQinv      = Qinv - Qinvi;
302   dibl_corr    = ( 1.0 - FF0 ) * ( 1.0 - Fsatq ) * qi * dQinv;
303   qd       = qd - dibl_corr;
304
305
306   //Inversion charge partitioning to terminals s and d
307   Qinvs      = type * Leff * (( 1 + dir ) * qs + ( 1 - dir ) *
        qd)/ 2.0;
308   Qinvd      = type * Leff * (( 1 - dir ) * qs + ( 1 + dir ) *
        qd)/ 2.0;
309
310
311   //Outer fringing capacitance
312   Qsov       = Cofs * ( V(g) - V(si) );
313   Qdov       = Cofd * ( V(g) - V(di) );
314
315
316   //Inner fringing capacitance
317   Vt0x       = Vt0 + gamma * ( sqrt( abs( phib - type * ( V(b)
        - V(si) ))) - sqrt(phib));
318   Vt0y       = Vt0 + gamma * ( sqrt( abs( phib - type * ( V(b)
        - V(di) ))) - sqrt(phib));
319   Fs_arg       = ( Vgsraw - ( Vt0x - Vdsi * delta * Fsat ) +
       aphit * 0.5 )/ ( 1.1 * nphit );
320   if (Fs_arg <= `LARGE_VALUE) begin
321     Fs     = 1.0 + exp( Fs_arg );
322     FFx    = Vgsraw - nphit * ln( Fs );
323   end
324   else begin
325     Fs     = 0.0;                  //  Not used
326     FFx    = Vgsraw - nphit * Fs_arg;
327   end
328   Fd_arg       = ( Vgdraw - ( Vt0y - Vdsi * delta * Fsat ) +
       aphit * 0.5 )/ ( 1.1 * nphit );
329   if (Fd_arg <= `LARGE_VALUE) begin
330     Fd     = 1.0 + exp( Fd_arg );
331     FFy    = Vgdraw - nphit * ln( Fd );
332   end
333   else begin
334     Fd     = 0.0;                  //  Not used
```

```
335     FFy    = Vgdraw - nphit * Fd_arg;
336   end
337   Qsif       = type * ( Cif + CC * Vgsraw ) * FFx;
338   Qdif       = type * ( Cif + CC * Vgdraw ) * FFy;
339
340
341   //Partitioned charge
342   Qs       = -W * ( Qinvs + Qsov + Qsif );            //  s-
      terminal charge
343   Qd       = -W * ( Qinvd + Qdov + Qdif );            //  d-
      terminal charge
344   Qg       = -( Qs + Qd + Qb );             //  g-terminal
      charge
345
346
347   //Sub-circuit initialization
348       I(di,si)     <+  type * dir * Id;
349       I(d,di)      <+  ( V(d) - V(di) )/ Rd;
350       I(si,s)      <+  ( V(si) - V(s) )/ Rs;
351
352     I(si,b)      <+  ddt( Qs );               //  charge term:
      node si to node b
353   I(di,b)      <+  ddt( Qd );               //  charge term:
      node di to node b
354   I(g,b)       <+  ddt( Qg );               //  charge term:
      node g to node b
355
356     //****** units of Qg and length are Coul and cm,
      respectively. V has unit of uCoul/cm^2 ******//
357     //added by Muhammad A. Wahab of Purdue University on May
      10, 2015
358     V(qg_as_v)       <+  Qg/W/Lgdr*1e6;
      // 1e6 is used for unit conversion of charge: Coul to
      uCoul
359
360
361 end
362 endmodule
```

Listing A.2: MOSFET virtual–source based self–consistent transport/capacitance model in Verilog–A description. Reference to [29].

Appendix B

Python scripts for the array management

B.1 Operations script

```
#! /usr/bin/env python
import sys
#from idlelib.colorizer import prog
#from anaconda_navigator.utils.encoding import write

#parameters
t0 = 5e-9
t_rise = t0/100
Vdd = 1
array_dim = 8
V_high=500e-3
hex_or_bin=1
operation = str(sys.argv[1])
cycle = int(sys.argv[4])
duration = int(sys.argv[5])
word = str(sys.argv[3])
col = int(sys.argv[2])
time_offset=int(sys.argv[6])

def array_op(operation, word, col, cycle, duration,
   time_offset):

    if operation == "prog":
        if hex_or_bin==0:
            #convert hex to bin
            len_word=4*len(word)
```

```
26              word=int(word,16)
27              word=bin(word)
28              word=word[2:].zfill(len_word)
29          else:
30              len_word=len(word)
31          #open files
32          file_SL=[]
33          file_WL=[]
34          for j in range(len_word):
35              file_SL.append(open(("SL_" + str(j) + ".csv"), 'a
    '))
36              file_WL.append(open(("WL_" + str(j) + ".csv"), 'a
    '))
37
38          #initialize
39          if time_offset==0:
40              for j in range(len_word):
41                  file_WL[j].write('0' + " " + '0' + "\n")
42                  file_SL[j].write('0' + " " + '0' + "\n")
43
44          t=1+time_offset
45          while t<=duration:
46              # WL now
47              for wl in range(len_word):
48                  if wl==col and t==cycle:
49                      file_WL[wl].write(str((t-1)*t0 + t_rise) +
     " " + '3' + "\n")
50                      file_WL[wl].write(str(t*t0) + " " + '3' +
    "\n")
51                  else:
52                      file_WL[wl].write(str((t-1)*t0 + t_rise)
    + " " + '0' + "\n")
53                      file_WL[wl].write(str(t*t0) + " " + '0' +
     "\n")
54              # SL now
55              for sl in range(len_word):
56                  if t==cycle:
57                      if int(word[sl])== 0:
58                          file_SL[sl].write(str((t-1)*t0 +
    t_rise) + " " + '3' + "\n")
59                          file_SL[sl].write(str(t*t0) + " " + '
    3' + "\n")
60                      elif int(word[sl])== 1:
61                          file_SL[sl].write(str((t-1)*t0 +
    t_rise) + " " + '0' + "\n")
```

```
                        file_SL[sl].write(str(t*t0) + " " + '0' + "\n")
                else:
                    file_SL[sl].write(str((t-1)*t0 + t_rise) + " " + '0' + "\n")
                    file_SL[sl].write(str(t*t0) + " " + '0' + "\n")

            t=t+1
        #close files now
        for j in range(len_word):
            file_WL[j].close()
            file_SL[j].close()

    if operation == "erase":
        #open files
        file_SL=[]
        file_WL=[]
        for j in range(array_dim):
            file_SL.append(open(("SL_" + str(j) + ".csv"), 'a'))
            file_WL.append(open(("WL_" + str(j) + ".csv"), 'a'))

        #initialize
        if time_offset==0:
            for j in range(array_dim):
                file_WL[j].write('0' + " " + '0' + "\n")
                file_SL[j].write('0' + " " + '0' + "\n")

        t=1+time_offset
        while t<=duration:
            # WL now
            for wl in range(array_dim):
               if wl==col and t==cycle:
                    file_WL[wl].write(str((t-1)*t0 + t_rise) + " " + '-5' + "\n")
                    file_WL[wl].write(str(t*t0) + " " + '-5' + "\n")
                else:
                    file_WL[wl].write(str((t-1)*t0 + t_rise) + " " + '0' + "\n")
                    file_WL[wl].write(str(t*t0) + " " + '0' + "\n")
            #SL now
            for sl in range(array_dim):
```

```
99                  file_SL[sl].write(str((t-1)*t0 + t_rise) + "
    " + '0' + "\n")
100                 file_SL[sl].write(str(t*t0) + " " + '0' + "\n
    ")
101             t=t+1
102         #close files
103         for j in range(array_dim):
104             file_WL[j].close()
105             file_SL[j].close()
106
107     if operation == "read":
108         #open files
109         file_Sel=[]
110         file_Sel_ref=open("Sel_ref.csv", 'a')
111         for j in range(array_dim):
112             file_Sel.append(open(("Sel_" + str(j) + ".csv"),
    'a'))
113
114         #initialize
115         if time_offset==0:
116             file_Sel_ref.write('0' + " " + '0' + "\n")
117             for j in range(array_dim):
118                 file_Sel[j].write('0' + " " + '0' + "\n")
119
120         t=1+time_offset
121         while t<=duration:
122             #Sel_ref now
123             if t==cycle:
124                 file_Sel_ref.write(str((t-1)*t0 + t_rise) + "
     " + str(V_high) + "\n")
125                 file_Sel_ref.write(str(t*t0) + " " + str(
    V_high) + "\n")
126             else:
127                 file_Sel_ref.write(str((t-1)*t0 + t_rise) + "
     " + '0' + "\n")
128                 file_Sel_ref.write(str(t*t0) + " " + '0' + "\
    n")
129             #Sel now
130             for sel in range(array_dim):
131                 if sel==col and t==cycle:
132                     file_Sel[sel].write(str((t-1)*t0 + t_rise
    ) + " " + str(V_high) + "\n")
133                     file_Sel[sel].write(str(t*t0) + " " + str
    (V_high) + "\n")
134                 else:
```

```
                    file_Sel[sel].write(str((t-1)*t0 + t_rise) + " " + '0' + "\n")
                    file_Sel[sel].write(str(t*t0) + " " + '0' + "\n")

            t=t+1

        #close files
        file_Sel_ref.close()
        for j in range(array_dim):
            file_Sel[j].close()

array_op(operation, word, col, cycle, duration, time_offset)
```

Listing B.1: Python script for generating the drivers of the memory in case of writing and reading.

B.2 Signals script

```
#! /usr/bin/env python
import sys

#parameters
t0=5e-9
t_rise=t0/100
Vdd=1
array_dim=8
Vclk_low=500e-3
V_high=500e-3
#variables
signal=str(sys.argv[1])
start=int(sys.argv[2])
stop=int(sys.argv[3])
duration=int(sys.argv[4])
time_offset=int(sys.argv[5])

def signal_op(signal, start, stop, duration, time_offset):

    if signal=="WL_ref":
        #open file
        file_wlref=open("WL_ref.csv", 'a')
        if time_offset==0:
            #initialize
            file_wlref.write('0' + " " + '0' + "\n")

        t=1+time_offset
```

```
28          while t<=duration:
29              if t>=start and t<=stop:
30                  file_wlref.write(str((t-1)*t0 +t_rise) + " "
    + '3' + "\n")
31                  file_wlref.write(str(t*t0) + " " + '3' + "\n"
    )
32              else:
33                  file_wlref.write(str((t-1)*t0 +t_rise) + " "
    + '0' + "\n")
34                  file_wlref.write(str(t*t0) + " " + '0' + "\n"
    )
35              t=t+1
36
37          #close file
38          file_wlref.close()
39
40      if signal=="clk":
41          #open file
42          file_clk=open("clk.csv", 'a')
43          if time_offset==0:
44              #initialize
45              file_clk.write('0' + " " + str(Vdd) + "\n")
46
47          t=1+time_offset
48          while t<=duration:
49              if t>=start and t<=stop:
50                  file_clk.write(str((t-1)*t0 +t_rise) + " " +
    str(Vclk_low) + "\n")
51                  file_clk.write(str(t*t0) + " " + str(Vclk_low
    ) + "\n")
52              else:
53                  file_clk.write(str((t-1)*t0 +t_rise) + " " +
    str(Vdd) + "\n")
54                  file_clk.write(str(t*t0) + " " + str(Vdd) + "
    \n")
55              t=t+1
56
57          #close file
58          file_clk.close()
59
60 signal_op(signal, start, stop, duration, time_offset)
```

Listing B.2: Python script for generating the signals of the sense amplifier in case of reading.

Bibliography

[1] G. Santoro, *Exploring New Computing Paradigms*, PhD thesis, Politecnico di Torino, 2019.

[2] N. Piano, *DExIMA: a Design Explorer for In-Memory Architectures*, Tesi di Laurea Magistrale, Politecnico di Torino, 2019.

[3] U. Casale, *Programmable LiM: a Modular and Reconfigurable Approach to the Logic in Memory*, Tesi di Laurea Magistrale, Politecnico di Torino, 2020.

[4] D. Ielmini,H.-S. Philip Wong, *In-memory computing with resistive switching devices*, Nature Electronics, 2018.

[5] J.Joshua Yang, Dmitri B. Strukov, Duncan R. Stewart, *Memristive devices for computing*, nature technology, 2012.

[6] J. Borghetti, G. S. Snider, P. J. Kuekes, J. J. Yang, D. R. Stewart, R. S. Williams, *'Memristive' switches enable 'stateful' logic operations via material implication*, Nature Letters, 2010.

[7] Simone Raoux, Wojciech Wełnic, Daniele Ielmini, *Phase Change Materials and Their Application to Nonvolatile Memories*, Chemical Reviews, 2010.

[8] MRAM-Info: the MRAM experts, *SOT-MRAM: Introduction and market status*, 19 February 2019.

[9] MRAM-Info: the MRAM experts, *STT-MRAM: Introduction and market status*, 19 February 2019.

[10] G. Prenat, K. Jabeur, G. Di Pendina, O. Boulle, G. Gaudin, *Beyond STT-MRAM, Spin Orbit Torque RAM SOT-MRAM for High Speed and High Reliability Applications*, Springer International Publishing Switzerland, 2015.

[11] P. Gambardella, *Introduction to spin torques and spin-orbit torques in metal layers*, Department of Materials, ETH Zurich, Switzerland, 2015.

[12] H. Kimura, T. Hanyu, M. Kameyama, Y Fujimori, T. Nakamura, H. Takasu, *Complementary Ferroelectric-Capacitor Logic for Low-Power*

Logic-in-Memory VLSI, IEEE Journal of Solid-State Circuits, 2004.

[13] Alex P. James, Linu R.cV. J. Francis, and Dinesh S. Kumar, *Resistive Threshold Logic*, IEEE transaction on very large scale integration (VLSI) systems, 2014.

[14] N. Talati, S. Gupta, P. Mane, S. Kvatinsky, *Logic Design within Memristive Memories Using Memristor Aided loGIC (MAGIC)*, IEEE transactions on nanotechnology, 2016.

[15] S. Kvatinsky, M. Ramadan, E. Friedman, A. Kolodny, *VTEAM: A general model for voltage-controlled memristors* IEEE Trans. Circuits Syst., 2015.

[16] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, K. N. Salama, *Memristor-based memory: The sneak paths problem and solutions*, Microelectronics Journal, 2013.

[17] J. Wang, H. Meng, Jian-P. Wang, *Programmable spintronics logic device based on a magnetic tunnel junction element*, Journal of Applied Physics, 2005.

[18] H. Kimura, T. Hanyu, M. Kameyama, Y. Fujimori, T. Nakamura, H. Takasu, *Complementary Ferroelectric-Capacitor Logic for Low-Power Logic-in-Memory VLSI*, IEEE Journal of Solid-State Circuits, 2004.

[19] T. Mikolajick, C. Dehm, W. Hartner, I. Kasko, M.J. Kastner, N. Nagel, M. Moert, C. Mazure, *FeRAM technology for high density applications*, Microelectronics Reliability, 2001.

[20] Myoung-J.Lee, Chang B. Lee, D. Lee, Seung R. Lee, M. Chang, Ji H. Hur, Young-B. Kim, Chang-J. Kim, David H. Seol, S. Seo, U-In Chung, In-K. Yoo, K. Kim, *A fast, high-endurance and scalable non-volatile memory device made from asymmetric Ta2 O5-x /TaO2-x bilayer structures*, Nature Materials, 2011.

[21] B. Choi, A. Torrezan, J. Strachan, P. G. Kotula, A. J. Lohn, M. J. Marinella, Z. Li, R. Williams, J. Yang, *High-Speed and Low-Energy Nitride Memristors*, Advanced Functional Materials Journal, 2016.

[22] E. T. Breyer, H. Mulaosmanovic, T. Mikolajick, S. Slesazeck, *Reconfigurable NAND/NOR logic gates in 28 nm HKMG and 22 nm FD-SOI FeFET technology*, IEEE Journal of Solid-State Circuits, 2017.

[23] E. T. Breyer, H. Mulaosmanovic, S. Slesazeck, T. Mikolajick, *Demonstration of versatile nonvolatile logic gates in 28nm HKMG FeFET technology*, IEEE Journal of Solid-State Circuits, 2018.

[24] D. Reis, M. T. Niemer, X. S. Hu, *A Computing-in-Memory Engine for Searching on Homomorphically Encrypted Data*, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, 2019.

[25] Y. Long, D. Kim, E. Lee, P. Saha, B. A. Mudassar, X. She, A. I. Khan, S. Mukhopadyhyay, *A Ferroelectric FET-Based Processing-in-Memory Architecture for DNN Acceleration*, IEEE Journal on Exploratory Solid-State Computational Devices and Circuits, 2019.

[26] E. T. Breyer, H. Mulaosmanovic, J.Trommer, T. M. Dunkel, M. Trentzsch, S. Beyer, S. Slezazeck, T. Mikolajick, *Compact FeFET Circuit Building Blocks for Fast and Efficient Nonvolatile Logic-in-Memory*, Journal of the Electron Devices Society, 2020.

[27] X. Yin, A. Aziz, J. Nahas, S. Datta, S. Gupta, M. Niemier , X. S. Hu, *Exploiting Ferroelectric FETs for Low-Power Non-Volatile Logic-in-Memory Circuits*, Conference Paper, 2016.

[28] K. Ni, M. Jerry, J. A. Smith, S. Datta, *A Circuit Compatible Accurate Compact Model for Ferroelectric-FETs*, Symposium on VLSI Technology Digest of Technical Papers, 2018.

[29] M. A. Wahab, M. A. Alam, *A Verilog-A Compact Model for Negative Capacitance FET*, Purdue University, 2016.

[30] F. Zahoor, T.Z.A. Zulkifli, F.A. Khanday, *Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc), Storage, Modelling and Applications*, Nanoscale Research Letters, 2020.

[31] T. Endoh, H. Honjo, *A recent Progress of Spintronics Devices for Integrated Circuit Applications*, Journal of Low Power Electronics and Applications, 2018.

[32] A. Aziz, S. Ghosh, S. Datta, S. K. Gupta, *Physics-Based Circuit-Compatible SPICE Model for Ferroelectric Transistors*, IEEE Electron Device Letters, Vol. 37, No. 6, June 2016.

[33] A. G. Maslovskayaa, L. I. Moroza, A. Yu Chebotarevbc, A. E. Kovtanyukbc, *Theoretical and numerical analysis of the Landau–Khalatnikov model of ferroelectric hysteresis*, Communications in Nonlinear Science and Numerical Simulation, 2021.

[34] Y.-H. Chen, T. Krishna, J. Emer, V. Eyeriss, *An energy-efficient reconfigurable accelerator for deep convolutional neural networks*, IEEE J. Solid-State Circuits, 2017.

[35] N. Jouppi et al, *In–datacenter performance analysis of a tensor processing unit*, Proc. 44th Int. Symp. Comp. Architecture (ISCA), 2017.

[36] J. T. Pawlowski, *Hybrid memory cube (HMC)*, IEEE Hot Chips 23 Symp., 2011.

[37] D. U. Lee et al, *A 1.2 V 8Gb 8-channel 128GB/s high–bandwidth memory (HBM) stacked DRAM with effective microbump I/O test methods using*

29 nm process and TSV, IEEE Int. Solid-State Circuits Conf. Digest Tech. Papers, 2014.

[38] M-F. Chang, S-S. Sheu, K-F. Lin, C-W. Wu, C-C. Kuo, P-F. Chiu, Y-S. Yang, Y-S. Chen, H-Y. Lee, C-H. Lien, F. T. Chen, K-L. Su, T-K. Ku, M-J. Kao, M.-J. Tsa, *High-Speed 7.2-ns Read-Write Random Access4-Mb Embedded Resistive RAM (ReRAM) MacroUsing Process–Variation–Tolerant Current–ModeRead Schemes*, IEEE JOURNAL OF SOLID-STATE CIRCUITS, 2013.

[39] J. Cooke, *Introduction to Flash Memory (T1A)*, Micron Technology, Inc., 2008.

[40] N. Ohtsuka, S. Tanaka, J. Miyamoto, S. Saito, S. Atsumi, K. Imamiya, K. Yoshikawa, N. Matsukawa, S. Mori, N. Arai, T. Shinagawa, Y. Kaneko, J. Matsunaga, T. IIzuka, *A 4-Mbit CMOS EPROM*, IEEE JOURNAL OF SOLID-STATE CIRCUITS, 1987.

[41] D. Baderna, A. Cabrini, G. De Sandre, F. De Santis, M. Pasotti, A. Rossini, G. Torelli, *A 1.2 V Sense Amplifier for High-Performance Embeddable NOR Flash Memories*, STMicroelectronics, 2005.

[42] D. Arora, A. K. Gundu, M. S. Hashmi, *A High Speed Low Voltage Latch Type Sense Amplifier for Non-Volatile Memory*, Indraprastha Institute of Information Technology, Delhi, 2016.

[43] L. Jiang et al., *A low-voltage sense amplifier for high-performance embedded flash memory*, SEMICONDUCTOR INTEGRATED CIRCUITS, 2010.

[44] J. K. Yadav, *Sense Amplifier for Flash Memories: Architectural Exploration and Optimal Solution*, Indraprastha Institute of Information Technology, 2015.

[45] UMBC Maryland, *Liberty Timing File (LIB)*, Advanced VLSI Design.