POLITECNICO DI TORINO

Master course in Electronics Engineering

Master degree thesis

Spice-level implementation and evaluation of the Logic-in-Memory paradigm

Supervisors
Prof. Mariagrazia Graziano
Prof. Marco Vacca
Prof. Maurizio Zamboni

Candidate
Lorenzo De Carlo Chimienti

Academic Year 2022-2023

Summary

In recent years, the Logic-in-Memory (LiM) paradigm has become a widely explored topic. It is an architectural solution aimed at solving the Von Neumann bottleneck problem, which arises from the existing performance gap between CPU and memory. LiM application may be achieved by placing simple computational elements near or inside the cell of a memory array. In this way, data is locally computed inside the memory itself, leading to faster and less energy-expensive solutions. Doing so, memory arrays become larger to accomodate the additional transistors and the proper memory behavior have to be assured.
This thesis explores different LiM cells (in-cell computation) which implement basic bitwise logic operations (i.e. AND, OR and XOR) between the memorized content and an external input signal. The starting point for the design of these memory cells is the 6-transistors (6T) cell, commonly employed in Static Random Access Memory (SRAM) arrays.
All the used electronic elements are handled at Spice-level, through the manipulation of netlist files. To this end, taking advantage of Python language and of the OCEAN (Open Command Environment for ANalysis) language provided by Cadence, is a crucial point. Thanks to this powerful tool, it is possible to create SRAM and LiM arrays, and run simulations without the need for a graphical user interface (GUI), speeding up the whole process.
The 6T cell is sized after transient analyses, which are performed to test the basic operation that are executed by a memory cell (i.e. writing, reading and data retaining). Tests are carried out exploiting some peripheral circuitry needed to carry out the operations. These circuital components are sized to properly work with the designed cells.
The logic functions inside the LiM circuits belong to the Dynamic Logic family. Only the pull-down network is needed, in addition to two transistors, one for the output precharge and one footer. This logic family, compared to the Static one, reduces area occupation and delays, making it a suitable choice.
The new circuits are tested in a similar way as the standard one previously mentioned. For each LiM element, the combinatorial operation must also

be taken into account when performing tests to assure the proper functioning.
An integrated circuit (IC) layout of the memory cell is realized to provide accurate area and performance evaluation including parasitic contributions (R and C) in the Spice models. In this way it is possible to perform simulations useful for testing the designed cells. Different layouts for the same circuit are drawn, in order to understand which configuration grants a lower parasitic contribution.
Furthermore, an enhanced version of the XOR LiM cell is designed and laid out, which provides the writing of the output bit into the storage node of the successive one. Such feature is a fundamental improvement which finally allows to have the manipulated data already inside the memory array.
The aim of this work is to compare the various cells in terms of area, dissipated energy and access delay, evaluating the feasibility of their implementation. As a final result, the development of memory cells which present a restrained increase in terms of energy consumption and access delay is achieved.

Contents

List of Tables 7

List of Figures 8

1 State of the Art 13
- 1.1 The Logic in Memory (LiM) paradigm . . . 13
  - 1.1.1 The von Neumann architecture . . . 13
  - 1.1.2 A Logic-in-Memory computing overview . . . 14
- 1.2 Static Random Access Memories (SRAMs) . . . 15
  - 1.2.1 Generic architecture . . . 15
  - 1.2.2 Memory cells . . . 15
  - 1.2.3 Address decoders . . . 19
  - 1.2.4 Precharge circuits . . . 20
  - 1.2.5 Sense amplifiers (SAs) . . . 21
  - 1.2.6 Write drivers . . . 22

2 The SRAM Array 27
- 2.1 Cadence Virtuoso . . . 27
- 2.2 Circuital components . . . 27
- 2.3 Memory cell characterization . . . 28
  - 2.3.1 Hold margin . . . 29
  - 2.3.2 Read margin . . . 29
  - 2.3.3 Write margin . . . 31
- 2.4 Memory cell layout . . . 32
  - 2.4.1 Description of the cell . . . 33
  - 2.4.2 DRC and LVS . . . 34
  - 2.4.3 PEX . . . 35
- 2.5 Peripheral circuits characterization . . . 35

2.5.1 Precharge circuit 36
2.5.2 Write driver 36
2.5.3 Sense amplifier 37
2.6 Simulations 38

3 The Logic Gates 43
3.1 The dynamic logic 43
3.2 De Morgan's laws 44
3.3 Gates design 44
3.3.1 Dynamic AND gate 45
3.3.2 Dynamic OR gate 45
3.3.3 Dynamic XOR gate 46

4 Logic-in-Memory Cells 47
4.1 Workflow 47
4.2 Schematics of the LiM cells 48
4.2.1 The LiM-AND cell 48
4.2.2 The LiM-OR cell 48
4.2.3 The LiM-XOR cell 49
4.2.4 The Write-Back XOR cell 50
4.3 Layouts of the LiM cells 53
4.3.1 The LiM-AND cell 53
4.3.2 The LiM-OR cell 55
4.3.3 The LiM-XOR cell 55
4.3.4 The Write-Back XOR cell 57
4.3.5 Parasitic Extraction from layouts 57

5 Post-layout simulations 59
5.1 AND-v1 60
5.2 OR-v1 63
5.3 AND-v2 65
5.4 OR-v2 68
5.5 XOR 71
5.6 XOR-WB-v1 75
5.7 XOR-WB-v2 77

6 Conclusions 81
6.1 Area 81
6.2 Delay 82
6.3 Consumption 82
6.4 Final considerations 83

List of Tables

2.1 Size of the 6T cell. 35
2.2 Delays for the 6T cell. 42
2.3 Energy consumption for the 6T cell. 42
2.4 Power consumption for the 6T cell. 42
4.1 Sizes of the layouts AND-v1 and AND-v2 for the LiM-AND cell. 54
4.2 Sizes of the layouts OR-v1 and OR-v2 for the LiM-OR cell. . 55
4.3 Sizes of the layout for the LiM-XOR cell and the layouts XOR-WB-v1 and XOR-WB-v2 for the LiM-XOR-WB cell. . 57
5.1 Delays of the LiM-AND, LiM-OR and LiM-XOR cells. . . . 73
5.2 Energy consumption of the LiM-AND, LiM-OR and LiM-XOR cells. 74
5.3 Power consumption of the LiM-AND, LiM-OR and LiM-XOR cells. 74
5.4 Delays of the LiM-XOR-WB cell. 79
5.5 Energy consumption of the LiM-XOR-WB cell. 79
5.6 Power consumption of the LiM-XOR-WB cell. 79
5.7 Delay of the write-back operation of the LiM-XOR-WB cell. 80
5.8 Energy consumption of the write-back operation of the LiM-XOR-WB cell. 80
5.9 Power consumption of the write-back operation of the LiM-XOR-WB cell. 80
6.1 Areas of all the designed LiM cells. 82
6.2 Delays of all the designed LiM cells. 82
6.3 Energy consumption of all the designed LiM cells. 83
6.4 Power consumption of all the designed LiM cells. 83

List of Figures

1.1 Logic-in-Memory paradigm. 14
1.2 Generic SRAM block scheme. 16
1.3 6T cell schematic. 16
1.4 8T cell schematic. 19
1.5 10T cell schematic. 19
1.6 Schematics of some of the possible precharge circuits. 22
1.7 Latch-type SA schematic. 23
1.8 Current mirror differential SA schematic. 23
1.9 Paired current mirror amplifier (PCMA) schematic. 24
1.10 PMOS cross-coupled amplifier (PCCA) schematic. 24
1.11 Transmission gates-based write driver schematic. 25
1.12 Pass transistor gates-based write driver schematic. 25
1.13 AND gates-based write driver schematic. 25
2.1 Schematic employed to characterize the 6T memory cell. . . 29
2.2 Inverters characteristic curves. 30
2.3 Schematic employed to test the read margin of the 6T cell. . 30
2.4 Internal nodes voltages as functions of the noise voltage during the read operation. 31
2.5 Internal nodes voltages as functions of the noise voltage during the read operation - zoom around the intersection of the two curves. 32
2.6 Schematic employed to test the write margin of the 6T cell. . 32
2.7 Internal nodes voltages as functions of the noise voltage during the write operation. 33
2.8 Layout of the 6T memory cell. 34
2.9 Write operation of a '0' on the Q node of a standard 6T cell. 38
2.10 Write operation of a '0' on the Q node of a standard 6T cell: detail around the rising edge of the PRE# signal. 38

2.11 Write operation of a '1' on the Q node of a standard 6T cell. 39
2.12 Write operation of a '1' on the Q node of a standard 6T cell: detail around the rising edge of the PRE# signal. 39
2.13 Read operation of a '0' on the BL/BL_B signals in a 6T SRAM array. 40
2.14 Read operation of a '0' on the BL/BL_B signals in a 6T SRAM array: detail around the rising edge of the PRE# signal. 40
2.15 Read operation of a '1' on the BL/BL_B signals in a 6T SRAM array. 41
2.16 Read operation of a '1' on the BL/BL_B signals in a 6T SRAM array: detail around the rising edge of the PRE# signal. 41
3.1 Dynamic NOR gate used to implement the logic function Q AND IN. 45
3.2 Dynamic NAND gate used to implement the logic function Q OR IN. 46
3.3 Dynamic XOR gate used to implement the logic function Q XOR IN. 46
4.1 Workflow followed for each LiM cell. 48
4.2 Schematic of the LiM-AND cell. 49
4.3 Schematic of the LiM-OR cell. 50
4.4 Schematic of the LiM-XOR cell. 51
4.5 Schematic of the LiM-XOR-WB cell. 52
4.6 Block scheme which depicts the write-back operation. 52
4.7 First layout of the LiM-AND cell: AND-v1. 53
4.8 Second layout of the LiM-AND cell: AND-v2. 54
4.9 First layout of the LiM-OR cell: OR-v1.. 55
4.10 Second layout of the LiM-OR cell: OR-v1. 56
4.11 Layout of the LiM-XOR memory cell. 56
4.12 First layout of the LiM-XOR-WB cell: XOR-WB-v1. 57
4.13 Second layout of the LiM-XOR-WB cell: XOR-WB-v2. . . . 58
5.1 Write operation of a '0' on the Q node of a AND-v1 cell. . . 60
5.2 Write operation of a '0' on the Q node of a AND-v1 cell: detail around the rising edge of the PRE# signal. 60
5.3 Write operation of a '1' on the Q node of a AND-v1 cell. . . 61

5.4 Write operation of a '1' on the Q node of a AND-v1 cell: detail around the rising edge of the PRE# signal. 61
5.5 Read operation of a '0' on the BL/BL_B signals in a AND-v1 array. 61
5.6 Read operation of a '0' on the BL/BL_B signals in a AND-v1 array: detail around the rising edge of the PRE# signal. 62
5.7 Read operation of a '0' on the BL/BL_B signals in a AND-v1 array. 62
5.8 Read operation of a '1' on the BL/BL_B signals in a AND-v1 array: detail around the rising edge of the PRE# signal. 62
5.9 Write operation of a '0' on the Q node of a OR-v1 cell. . . . 63
5.10 Write operation of a '0' on the Q node of a OR-v1 cell: detail around the rising edge of the PRE# signal. 63
5.11 Write operation of a '1' on the Q node of a OR-v1 cell. . . . 63
5.12 Write operation of a '1' on the Q node of a OR-v1 cell: detail around the rising edge of the PRE# signal. 64
5.13 Read operation of a '0' on the BL/BL_B signals in a OR-v1 array. 64
5.14 Read operation of a '0' on the BL/BL_B signals in a OR-v1 array: detail around the rising edge of the PRE# signal. . . 64
5.15 Read operation of a '1' on the BL/BL_B signals in a OR-v1 array. 65
5.16 Read operation of a '1' on the BL/BL_B signals in a OR-v1 array: detail around the rising edge of the PRE# signal. . . 65
5.17 Write operation of a '0' on the Q node of a AND-v2 cell. . . 65
5.18 Write operation of a '0' on the Q node of a AND-v2 cell: detail around the rising edge of the PRE# signal. 66
5.19 Write operation of a '1' on the Q node of a AND-v2 cell. . . 66
5.20 Write operation of a '1' on the Q node of a AND-v2 cell: detail around the rising edge of the PRE# signal. 66
5.21 Read operation of a '0' on the BL/BL_B signals in a AND-v2 array. 67
5.22 Read operation of a '0' on the BL/BL_B signals in a AND-v2 array: detail around the rising edge of the PRE# signal. 67
5.23 Read operation of a '0' on the BL/BL_B signals in a AND-v2 array. 67

5.24 Read operation of a '1' on the BL/BL_B signals in a AND-v2 array: detail around the rising edge of the PRE# signal. 68
5.25 Write operation of a '0' on the Q node of a OR-v2 cell. . . . 68
5.26 Write operation of a '0' on the Q node of a OR-v2 cell: detail around the rising edge of the PRE# signal. 68
5.27 Write operation of a '1' on the Q node of a OR-v2 cell. . . . 69
5.28 Write operation of a '1' on the Q node of a OR-v2 cell: detail around the rising edge of the PRE# signal. 69
5.29 Read operation of a '0' on the BL/BL_B signals in a OR-v2 array. 69
5.30 Read operation of a '0' on the BL/BL_B signals in a OR-v2 array: detail around the rising edge of the PRE# signal. . . 70
5.31 Read operation of a '1' on the BL/BL_B signals in a OR-v2 array. 70
5.32 Read operation of a '1' on the BL/BL_B signals in a OR-v2 array: detail around the rising edge of the PRE# signal. . . 70
5.33 Write operation of a '0' on the Q node of a XOR cell. 71
5.34 Write operation of a '0' on the Q node of a XOR cell: detail around the rising edge of the PRE# signal. 71
5.35 Write operation of a '1' on the Q node of a XOR cell. 71
5.36 Write operation of a '1' on the Q node of a XOR cell: detail around the rising edge of the PRE# signal. 72
5.37 Read operation of a '0' on the BL/BL_B signals in a XOR array. 72
5.38 Read operation of a '0' on the BL/BL_B signals in a XOR array: detail around the rising edge of the PRE# signal. . . 72
5.39 Read operation of a '1' on the BL/BL_B signals in a XOR array. 73
5.40 Read operation of a '1' on the BL/BL_B signals in a XOR array: detail around the rising edge of the PRE# signal. . . 73
5.41 Write operation of a '0' on the Q node of a XOR-WB-v1 cell. 75
5.42 Write operation of a '1' on the Q node of a XOR-WB-v1 cell. 75
5.43 Read operation of a '0' on the BL/BL_B signals in a XOR-WB-v1 array. 75
5.44 Read operation of a '1' on the BL/BL_B signals in a XOR-WB-v1 array. 76

5.45 Writing the output of Cell 0 into the node Q of Cell 1: case in which a '0' is written back. 76
5.46 Writing the output of Cell 0 into the node Q of Cell 1: case in which a '1' is written back. 76
5.47 Write operation of a '0' on the Q node of a XOR-WB-v1 cell. 77
5.48 Write operation of a '1' on the Q node of a XOR-WB-v1 cell. 77
5.49 Read operation of a '0' on the BL/BL_B signals in a XOR-WB-v1 array. 77
5.50 Read operation of a '1' on the BL/BL_B signals in a XOR-WB-v1 array. 78
5.51 Writing the output of Cell 0 into the node Q of Cell 1: case in which a '0' is written back. 78
5.52 Writing the output of Cell 0 into the node Q of Cell 1: case in which a '1' is written back. 78

Chapter 1

State of the Art

1.1 The Logic in Memory (LiM) paradigm

1.1.1 The von Neumann architecture

In order to understand the advantages brought by the Logic in Memory paradigm is important to know what a von Neumann architecture is.
The von Neumann architecture is a computer architecture composed of a precise set of interconnected parts: a control unit (CU), an arithmetic-logic unit (ALU) (where CU and ALU compose the central processing unit - CPU), a memory unit, an input and an output [1].
One of the central features of this paradigm is the exchange of data between the memory unit and the ALU. Data are fetched from the memory, sent to the ALU and elaborated by means of arithmetic or logic operations. Then, the results of such operations are written back in the memory.
Two main problems have arisen, during the last decades, about the use of this kind of architectures. The first one is related to the fact that memories have not improved as fast as the logic, so data cannot be provided to the CPU with the fastest rate possible. This first issue is called *von Neumann bottleneck*. The second one is related to the power spent to access the memory, for both the reading and the writing operations. In fact, data has to be moved from the CPU to the memory (and viceversa), and this movement is responsible of a high percentage of the power consumption [2], [3].

1.1.2 A Logic-in-Memory computing overview

A way to partially overcome these issues, is the adoption of the so called *Logic-in-Memory* (LiM) paradigm, which consists on the implementation of logic elements near or inside memory ones.
It is possible to observe different approaches to this topic in the literature, and in [2] a classification is presented. Such a classification is made reasoning on how the memory is used for the computation.
The main typologies that have been identified are:

- Computation-near-Memory (CnM): the computation unit and the memory one are stacked. This approach is useful in order to reduce the consumption contribution from the interconnections, even though logic and memory are still two separate components as in the von Neumann architectures.

- Computation-in-Memory (CiM): the computation is performed by conveniently modified sense amplifiers. The memory array structure is not modified.

- Computation-with-Memory (CwM): memory intrinsically performs calculations, by employing a Look Up Table (LUT) and a Content Addressable Memory (CAM) to store inputs and outputs.

- Logic-in-Memory (LiM): simple logic is integrated directly inside the memory cell.

The cells developed in this thesis work belong to the last category of this classification. So, details on how these cells behave will be given in the later sections.

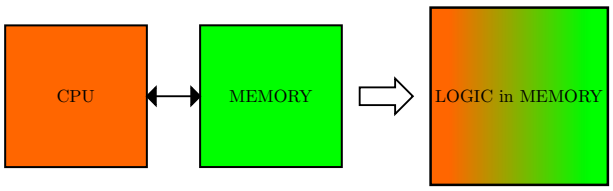

Figure 1.1. Logic-in-Memory paradigm.

1.2 Static Random Access Memories (SRAMs)

Static Random Access Memories represent one of the two main families of volatile memories (along with Dynamic RAMs). Being volatile, they require to be connected to the power supply in order to maintain the stored data after the use.
They are mainly employed in the realization of *caches*: memories in which are stored the most frequently required data. SRAMs replaced DRAMs as building blocks for the implementation of cache memories for two main reasons [4]:

- Due to their structure, they are faster w.r.t. DRAMs. This means that SRAMs are more recommended to be used near the microprocessor, because it is faster than any memory, and it is convenient that it accesses in a faster way at least the data that need to be processed more frequently.
- The technological process employed for SRAMs is the same as the one employed for the microprocessor logic. This implies that during the fabrication steps the same pieces of equipment (such as masks) used for the logic can be employed also for the memory realization.

1.2.1 Generic architecture

SRAMs consist of an array (organized in rows and columns) of memory cells capable of storing 1 bit and some peripheral circuitry, in which are usually identified the following main components: sense-amplifiers, address decoders, precharge circuits and write drivers. Each one of this circuits will be introduced in the following sections.

1.2.2 Memory cells

Single bit memory cells (bitcells) represent the basic unit of SRAMs. As long as the memory is powered up, each cell can be in one of the three possible states: *read*, *write*, *data retention*.
The most common implementation consists in the 6T cell. Its name comes from the fact that there are 6 MOS transistors in it. Referring to the scheme in Figure 1.3, they are respectively: 2 pull-up pMOS (Mpu1, Mpu2), 2 pull-down nMOS (Mpd1, Mpd2) and 2 nMOS pass-transistors (Mpt1, Mpt2).

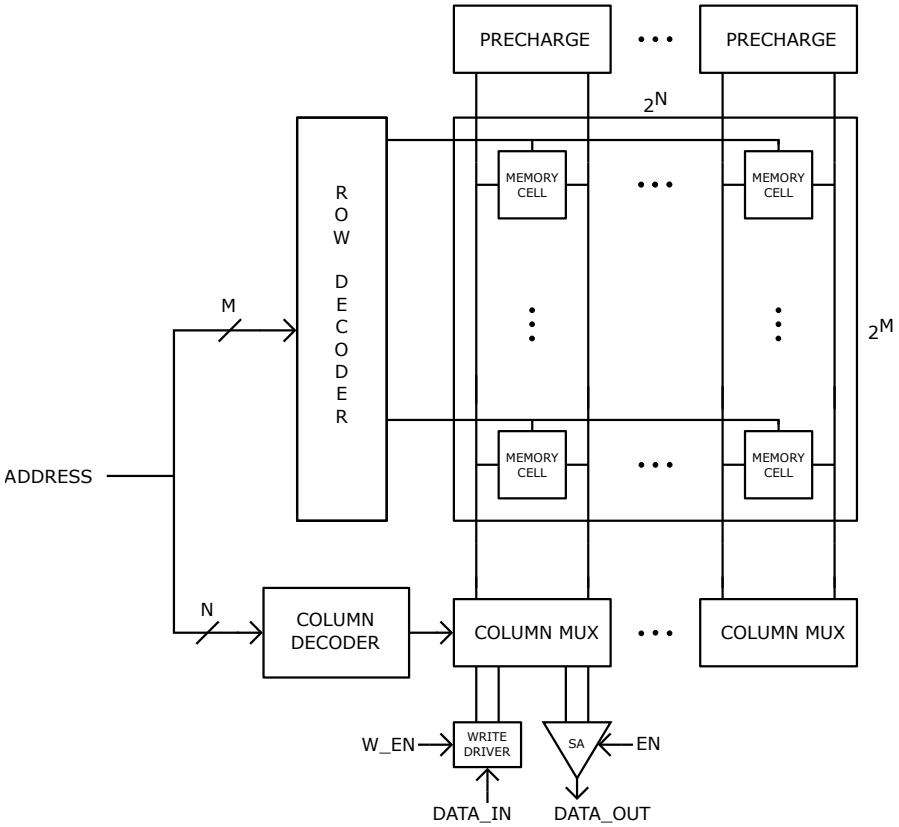

Figure 1.2. Generic SRAM block scheme.

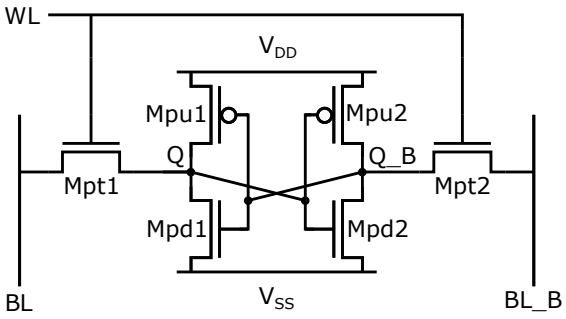

Figure 1.3. 6T cell schematic.

Mpu1, Mpu2, Mpd1 and Mpd2 realize two cross-connected inverters, which have the role of keeping the information. Mpt1 and Mpt2 are used to access the content of the cell, in order to perform both the read or the write operations. Every memory cell is accessed by means of a wordline

(WL) and a couple of bitlines (BL and BL_B), which respectively indicate the row and the column of each cell.

Static Noise Margin and 6T cell sizing

The dimension of the devices which compose a 6T cell must be carefully designed [5].
For this purpose, it is necessary to define an important parameter, the *Static Noise Margin (SNM)*: it is defined as the maximum noise voltage that can be applied to the internal node in order to not flip the content of the cell. Starting from this definition three different quantities can be distinguished. Each of these quantities plays a role in the design of the sizing of the devices. They are:

- Hold margin: SNM in absence of read or write operations.
- Read margin: SNM for reading. If we assume Q = '0' and Q_B = '1', during the read operation the currents in Mpd1 and Mpt1 can be equalized. In order to not alter the content of the cell, the voltage at node Q must be lower than the V_{th} of transistor Mpd2.
  Defining a value called **cell ratio** as

  $$r \equiv \frac{\beta_{pd1(2)}}{\beta_{pt1(2)}}$$

  and substituting it in

  $$I_{Dpd1(2)} = I_{Dpt1(2)} \tag{1.1}$$

  one obtains

  $$\mathrm{V}_Q = \tfrac{1}{r} \cdot (V_{DSATn} + r \cdot (V_{DD} - V_{thn}) - \sqrt{V_{DSATn}^2(1+r) \cdot r^2 \cdot (V_{DD} - V_{thn})^2} < V_{thn} \tag{1.2}$$

  In order to satisfy (1.2), **r** must be greater than 0. This implies that Mpd1 (Mpd2) has to be bigger than Mpt1 (Mpt2).
- Write margin: SNM for writing. A similar reasoning can be made for the write operation. If we assume to write a '1' in a cell storing a '0'... The solution is to write a '0' on the node Q_B, instead (and hence a

'1' on the node Q). By doing so, a current path between BL_B and V_{DD} is created: it means that the currents in Mpu2 and Mpt2 can be equalised.The voltage on the node Q_B must be pulled below the V_{th} of Mpd1 in order to let Mpu1 pull up the node Q.
In a similar fashion as before a new value called **pull-up ratio** can be defined

$$q \equiv \frac{\beta_{pu1(2)}}{\beta_{pt1(2)}}$$

and substituting it in

$$I_{Dpu2(1)} = I_{Dpt2(1)} \tag{1.3}$$

one obtains

$$\mathrm{V}_{Q\_B} = (V_{DD} - V_{thn}) - \sqrt{(V_{DD} - V_{thn})^2 - 2 \cdot \frac{\mu_p}{\mu_n} p \cdot ((V_{DD} - |V_{thp}|) \cdot V_{DSATp} - \frac{V_{DSATp}^2}{2})} < V_{thn} \tag{1.4}$$

In order to satisfy (1.4), **q** must be lower than 1. This implies that Mpt1 (Mpt2) has to be bigger than Mpu1 (Mpu2).

Variations

Starting from the standard 6T cell, some other SRAM cells have been designed. The following analyzed cells (8T and 10T) [6] consist in a variation of the 6T cell, where reading and writing operations occur on two different ports. This means that there is a distinction between the wordline used to access the cells for writing (W_WL) and the one used to access the cells for reading (R_WL). Also, another bitline devoted to the reading operation has been implemented (R_BL).
The schematics for these two cells are shown in Figure 1.4 and Figure 1.5.
The 8T cell does not introduce other advantages besides the isolated read-port. It actually introduces a new leakage current contribution due to the presence of an additional bitline.
The 10T cell manages to reduce this leakage contribution thanks to the presence of transistor Mnr. When R_WL of the cell under analysis is not selected, Mnr is off, adding another off transistor to the leakage current path (in addition to Mptr). This results in a lower leakage current contribution of the read port.

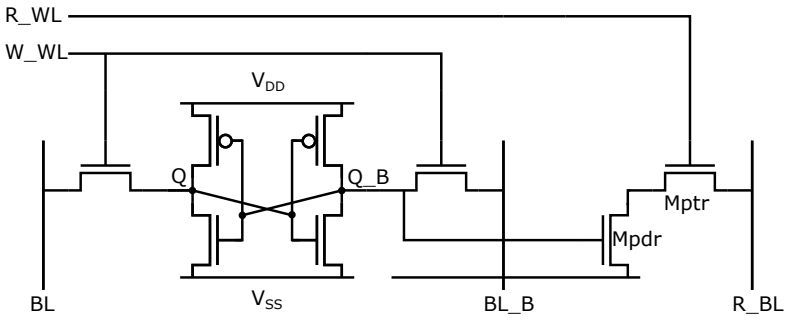

Figure 1.4. 8T cell schematic.

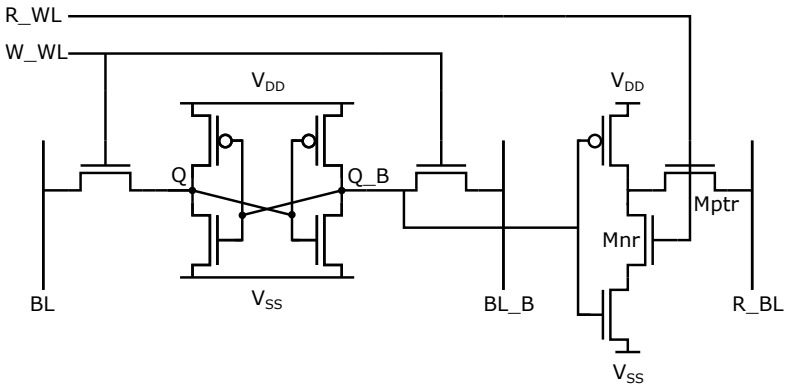

Figure 1.5. 10T cell schematic.

1.2.3 Address decoders

For large memory arrays, the use of decoders to access the different cells is fundamental in order to reduce the access time. Being 2^M the number of words and 2^N the parallelism of each word, the memory array would need 2^M and 2^N address lines for rows and columns respectively.
To have a more concrete idea of this concept, let us use a numerical example. If we assume to have a 32 Mb (2^{25} bits) memory with a parallelism of 32 bits ($2^N = 2^5$), we would end up with a number of rows $2^M = 2^{25-5} = 2^{20}$. So, in this particular example, the use of a binary decoder (n to 2^n) would decrease the number of needed row address lines from $1.05 \cdot 10^6$ to 20.
The simplest decoder implementation employs a collection of AND gates, whose inputs are the different possible combinations of the address bits(both true and complemented values).

Predecoding

One way to make a digital circuit more efficient, is to reduce its logical effort, a quantity which is defined as *the ratio of the input capacitance of a gate and the one of an inverter which delivers the same output current.* Since it is a linear function of the fan-in of a gate, finding a way to reduce the latter, can be a solution to the problem.
One common solution is called predecoding and consists in using more than one level of logic for the decoding operation. The employed gates are smaller and so present a lower fan-in, and by consequence a lower logical effort.
On the other hand, one disadvantage of this method is the need to use and distribute more wires w.r.t. the standard case [4].

Two-dimensional decoding

For what concerns flexibility, the memory can be accessed in a two-dimensional way, dividing the address bits in two subsets: one for selecting a column and the other one for selecting a row.
By adding a multiplexer, one sense amplifier can be shared by different s, ending up with power and area saving [4].

Sum-addressed decoders (SADs)

In some microprocessors instruction sets the address can be obtained by summing a base address and an offset value. In order to reduce the memory access time, sum-addressed decoders contains 2^M (number of wordlines) comparators, fed with the two strings, which check if the sum of *base+offset* is equal to the address of each wordline.
The check is reported to be faster than the actual sum computation, due to the fact that no carry propagation has to occur [7].

1.2.4 Precharge circuits

The bitline pair has to be precharged to V_{DD} before the reading operation takes place. Doing so, when a cell is accessed to be read, only one of the two bitlines changes its state, from '1' to '0'. This is because data inside SRAM cells are always stored by having one internal node to '1' and the other to '0', regardless of the fact that the datum is a '1' or a '0'.

Figure 1.6 shows some of the most common realizations of circuits responsible for the precharge of BL and BL_B [4].
The first reported circuit is the simplest implementation of a precharge circuit: it consists of a couple of diode-connected NMOS transistors, whose drains are connected to V_{DD} and sources to BL and BL_B. In this way the bitline pair is precharged to the value V_{DD} - V_{th}, where V_{th} is the threshold voltage of the devices. This solution is convenient in terms of complexity, but it does not allow to save power, since the circuit always tries to precharge the bitline pair, even during the write operation.
The second circuit implements also an equalization of the two bitline, through the addition of a PMOS transistor. This PMOS equalizes BL and BL_B when it is activated. This is done because the V_{th} of the two NMOS can differ, and so the two bitline would not be precharged to the same value.
The third circuit employs an NMOS to generate the V_{DD} - V_{th} level, but the bitline pair is precharged to this value through two PMOS transistors, which are activated along with the equalization device (by the same signal).
The last presented circuit consists in a variation of the previous one. The PMOS transistors connect BL and BL_B to V_{DD}, and no more to V_{DD} - V_{th}.

1.2.5 Sense amplifiers (SAs)

Sense amplifiers are analog circuits needed to speed up the memory reading process. The bitline pairs are precharged to '1' before the read operation; and since they present a large capacitive parasitic contribution (proportional to the number of cells that are connected to the said bitlines), the discharge operation that has to take place to write a '0' slows down the process.
So, SAs are employed to amplify a small voltage differential voltage that forms between the bitline and its complemented version, and convert it to a full swing voltage value. In this way, the time needed for reading decreases significantly.
Here are reported some of the possible different implementation for SAs [4]. All of them are activated when the enable (EN) signal is asserted
Figure 1.7 shows a latch-type SA. This amplifier consists in two cross-coupled inverters. It requires a small area to be implemented.
The next implementations to be presented are two similar ones. They are

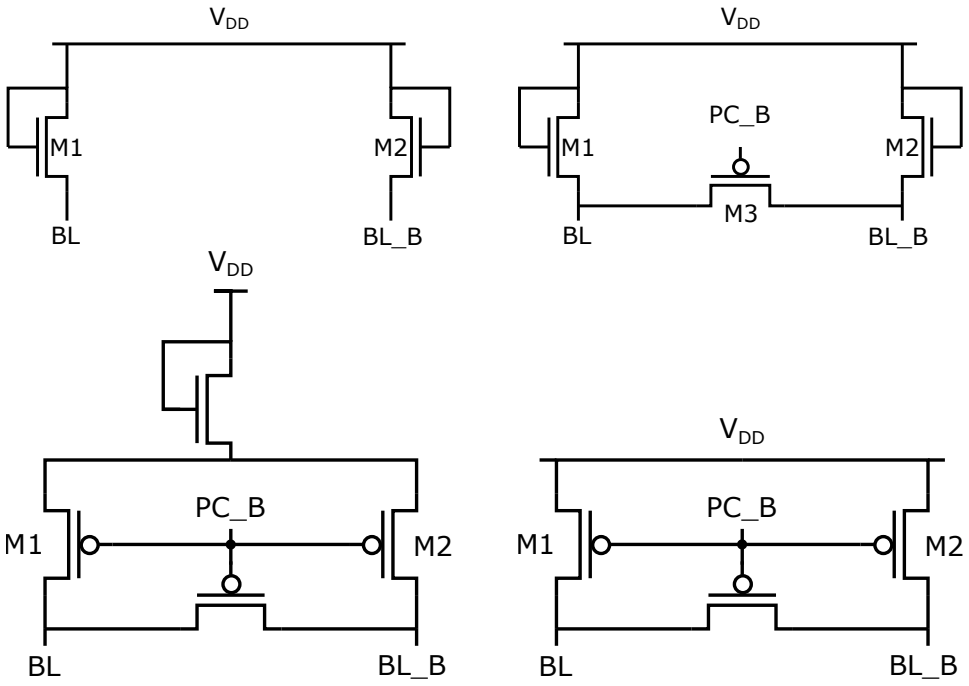

Figure 1.6. Schematics of some of the possible precharge circuits.

the current mirror differential SA (Figure 1.8) and the paired current mirror amplifier (PCMA) (Figure 1.9).
Being based on differential amplifiers, these versions of the circuit are useful in order to reject the common-mode component that may be present on the bitline and complemented bitline. They consist in an analog differential pairs whose load is a current-mirror.
When the amplifiers are enabled, a static current path between V_{DD} and V_{SS} is formed, introducing extra power dissipation.
The last SA presented is the PMOS cross-coupled amplifier (PCCA), shown in Figure 1.10. This implementation consumes less current w.r.t. the current mirror differential SA and the PCMA because there is no static current path formed.

1.2.6 Write drivers

The last peripheral circuits to be presented in this work, are the ones needed by the SRAM when data has to be written inside the memory cells. Such circuits are called write drivers. One write driver is connected to each

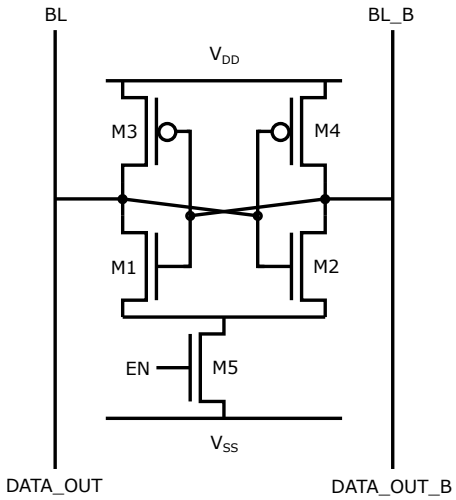

Figure 1.7. Latch-type SA schematic.

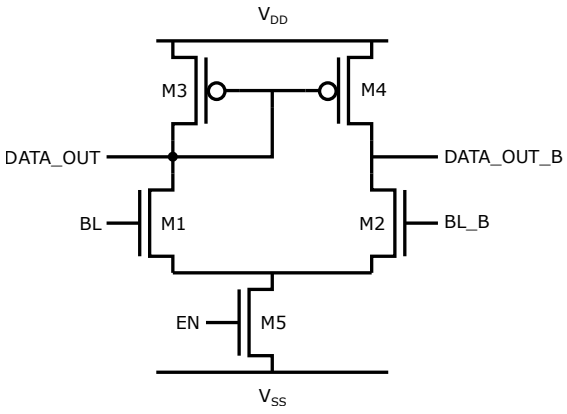

Figure 1.8. Current mirror differential SA schematic.

couple of bitlines, meaning that each SRAM array presents a total of 2^N (number of columns) of them.
The principle behind the different types of write drivers is the same: when the write enable (W_EN) signal is asserted, the value at the input of the circuit is written onto BL and its complemented value onto BL_B.
Some possible implementations are presented here [6]. In Figure 1.11 the elements that allow the driving of the bitlines are two transmission gates conveniently driven by the W_EN signal. The inverters I1 and I2 are responsible for the driving of respectively BL_B and BL.

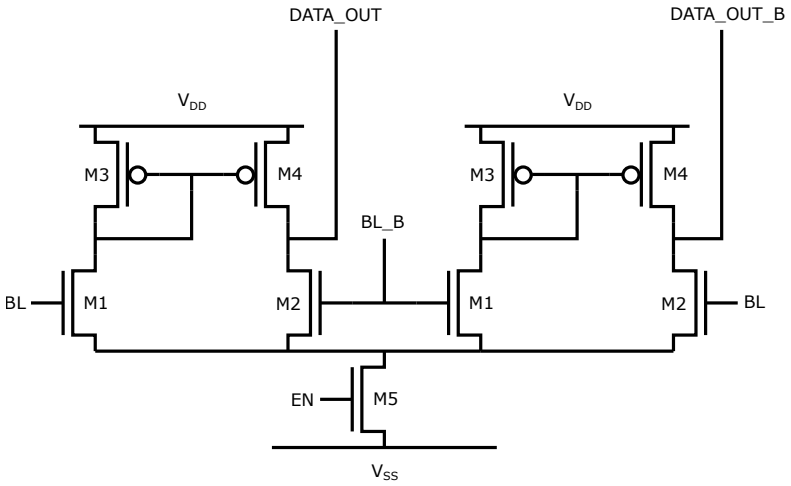

Figure 1.9. Paired current mirror amplifier (PCMA) schematic.

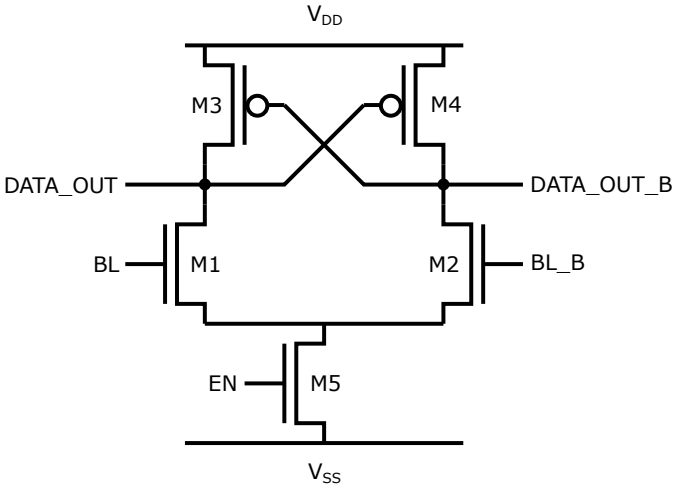

Figure 1.10. PMOS cross-coupled amplifier (PCCA) schematic.

The write drivers shown in Figure 1.12 and Figure 1.13 are two other possible implementations. They both rely on two AND gates, where in Figure 1.12 they are implemented with pass transistors (M1,M3 and M2,M4), while in Figure 1.13 there are two generic AND gates which drive M1 and M2.

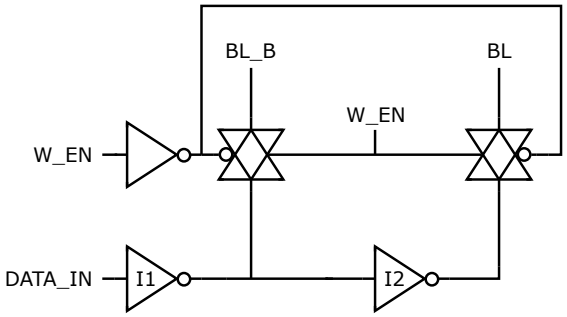

Figure 1.11. Transmission gates-based write driver schematic.

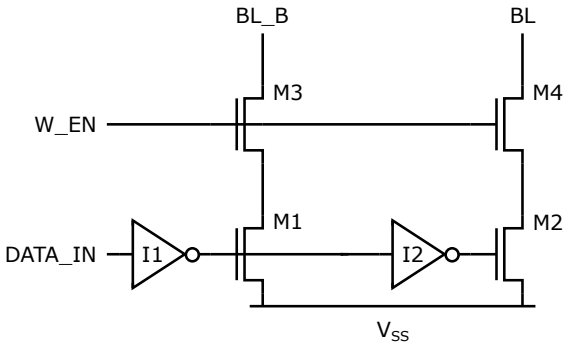

Figure 1.12. Pass transistor gates-based write driver schematic.

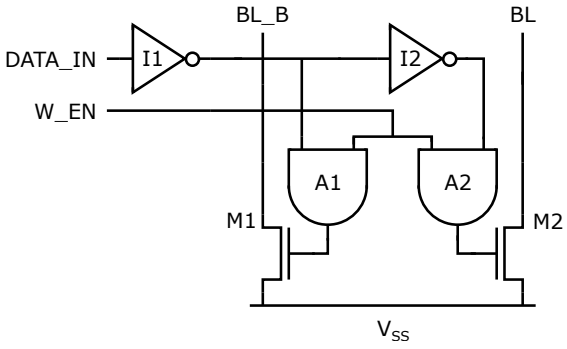

Figure 1.13. AND gates-based write driver schematic.

Chapter 2

The SRAM Array

2.1 Cadence Virtuoso

The first step of the work consists in implementing in *Cadence Virtuoso* the circuits which are present in the SRAM array seen in section 1.2.
Cadence Virtuoso, from now on simply referred as Virtuoso, is an Electronic Design Automation (EDA) tool; that in this work is employed to design both schematics, layouts and test benches, as it is explained later. To do so, the FreePDK45 design kit from the North Carolina State University (NCSU) is used. It contains the *NCSU_TechLib_FreePDK45* technology library, which provides all the resources needed to design circuits with a 45 nm process.

2.2 Circuital components

In this section is explained which implementation is chosen for each component, and the motivation for the choice.
The implemented circuits are the following:

- Memory cell: in the initial part of the work, the standard **6t cell** (Figure 1.3) is implemented. It is the simplest cell architecture and it is sufficient to characterize the peripheral circuitry in a correct way.
- Precharge circuit: for the bitline conditioning, the precharge circuit at the bottom right of Figure 1.6 is chosen. It is the simplest solution that presents both bitlines equalization, which is a desirable feature for the correct functioning of the component; and PMOS pass transistors, which is the best choice for passing V_{DD}.

- Sense amplifier: for the bitline sensing, the sense amplifier in Figure 1.7 is the simplest one. Since, the involved transistors cannot have minimum size, and since one SA have to be instantiated for each bitline pair, choosing a simple solution is beneficial in terms of both area and power dissipation.
- Write driver: for the writing operation, the driver in Figure 1.12 is realized. Among the considered implementation, this one is the less area-consuming one.

For what concerns the decoder, its function is beyond the aims of this work, so it is not implemented. As it is reported in a later section, the memory is accessed by directly enabling the wordline that has to be read or written.
The aim of this first step is to size all the MOS transistors that make up the above mentioned circuits. In order to do so, a schematic for each component is designed, as well as test bench schematics to study the behavior of such circuits, making it possible to size the transistors in the correct way.
As the schematics are created, all the MOS have parametric width (W), while their length (L) is left at its minimum value, which for the 45 nm technology used, is 50 nm.
All the schematic of this work present, as supply voltages, $V_{DD} = 1\,\mathrm{V}$ and $V_{SS} = 0\,\mathrm{V}$. The latter is simply referred to as *gnd*, the reference voltage.

2.3 Memory cell characterization

The characterization of the memory cell slightly differs from the one needed for the other components. In order to size the transistors, the Static Noise Margin defined in subsection 1.2.2 must be taken into account; along with the two important parameters defined in the same subsection: the cell ratio and the pull-up ratio. The aim of this step is to obtain acceptable values for the hold margin, the read margin and the write margin; in order to make the cell behave properly.
A quantity that plays a key role in this part of the characterization is the noise that can be present on both the internal nodes of the memory cell.
For this reason, the schematic used for the memory cell characterization is the one depicted in Figure 2.1.
The three simulations are performed by varying the W of the transistors, with the goal to satisfy both Equation 1.2 and Equation 1.4.

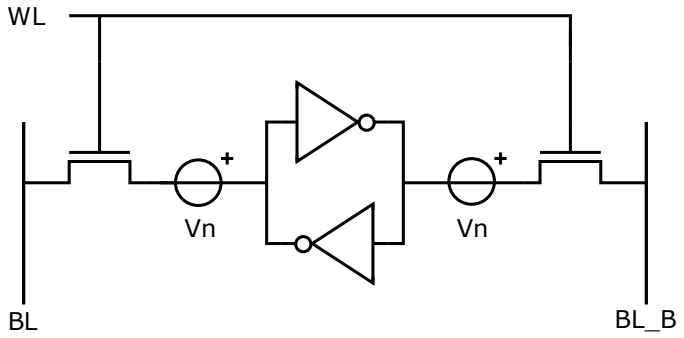

Figure 2.1. Schematic employed to characterize the 6T memory cell.

The reported results for the hold margin, the read margin and the write margin are the ones obtained with the final W values, which are the following ones:

- $Wn = 150\,\text{nm}$ - W of the two pull-down NMOS.
- $Wa = 100\,\text{nm}$ - W of the two access NMOS.
- $Wp = 90\,\text{nm}$ - W of the two pull-up PMOS.

2.3.1 Hold margin

For this test, it is sufficient to set the noise voltage $V_n = 0\,\text{V}$ (fig. x) and to plot the characteristic curves of the two inverter that compose the memory element. In Figure 2.2 is shown the so called "butterfly curve" because of the shape that may remind the one of a butterfly. The side of the minimum square that can be inscribed in the two sections of such graph, represents the SNM, in this case the hold margin. **immagine test hold**
From Figure 2.2 it is possible to measure the value of the hold margin, which is approximatively equal to 280 mV

2.3.2 Read margin

To simulate the read operation, both BL and BL_B are connected to V_{DD}. Assuming that the cell stores a '0' and a '1' on its internal nodes, the aim of the simulation is to see for which noise quantities the noise generators are able to write into the memory cell during when it is accessed for reading.

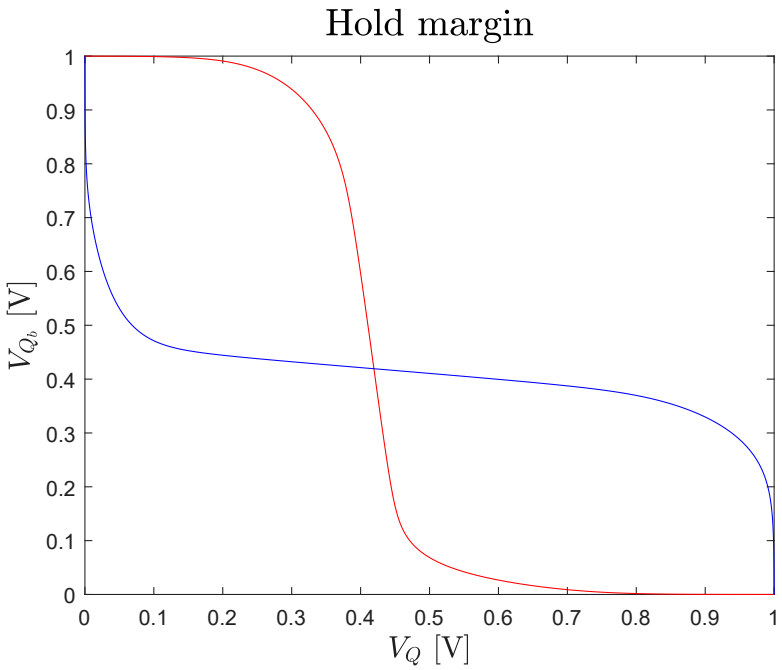

Figure 2.2. Inverters characteristic curves.

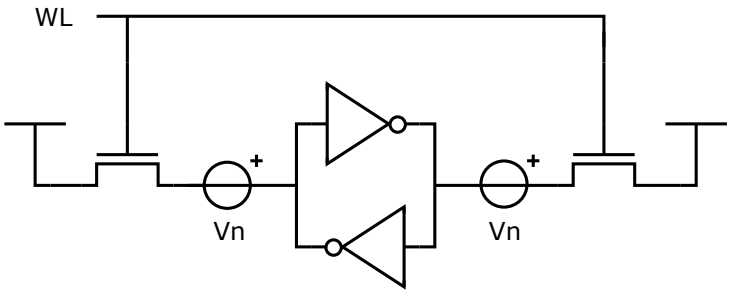

Figure 2.3. Schematic employed to test the read margin of the 6T cell.

In Figure 2.4 are depicted the voltages of the two memory cell internal nodes as functions of the noise voltage present at the input of the two inverters, during the read operation.
Since the two curves are very steep in the region around the intersection, Figure 2.5 shows a magnified version of such region. It is possible to notice that the curves intersect for $V_{noise} \simeq 142\,\text{mV}$, meaning that this value is the read margin of the memory cell.

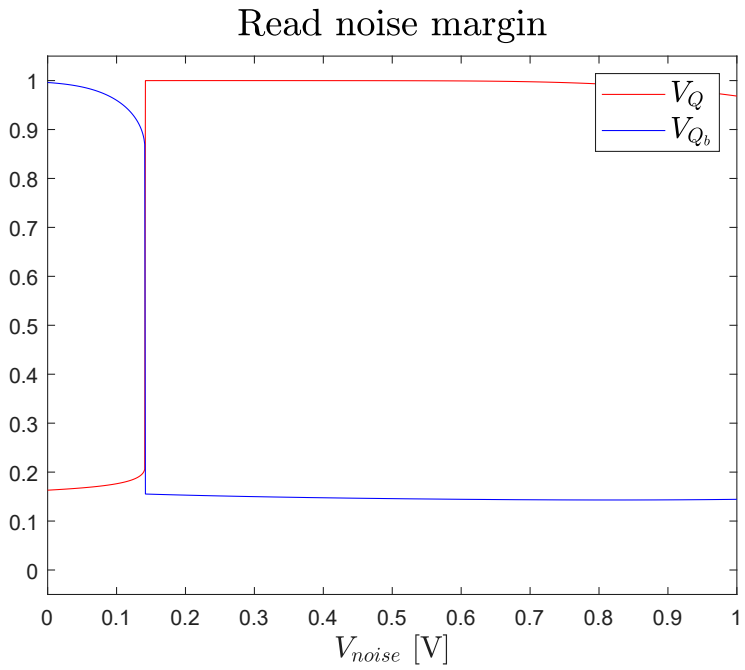

Figure 2.4. Internal nodes voltages as functions of the noise voltage during the read operation.

2.3.3 Write margin

To simulate the write operation, BL is connected to gnd, while BL_B is connected to V_{DD} also in this case. This is done because during writing, the write driver connects to gnd one of the two bitlines, depending on which data has to be written into the memory cell. Assuming to store a '0', the aim of the simulation is to see for which noise quantities the noise generators are able to alter this operation and write a '1' instead.

In Figure 2.7 are depicted the voltages of the two memory cell internal nodes as functions of the noise voltage present at the input of the two inverters, during the write operation.
In this simulation the intersection point is clearly visible without the need to zoom around it. The corresponding noise value is $V_{noise} \simeq 390\,\text{mV}$, representing the write margin of the memory cell.

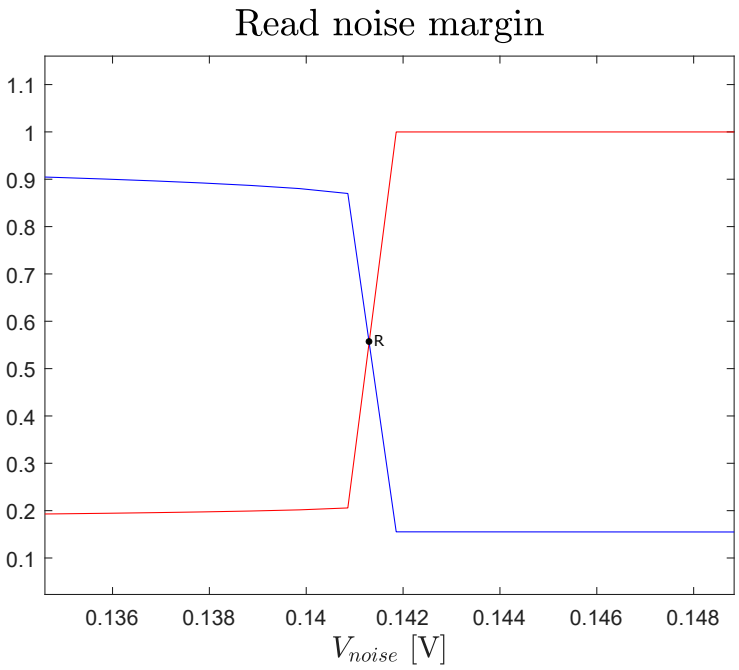

Figure 2.5. Internal nodes voltages as functions of the noise voltage during the read operation - zoom around the intersection of the two curves.

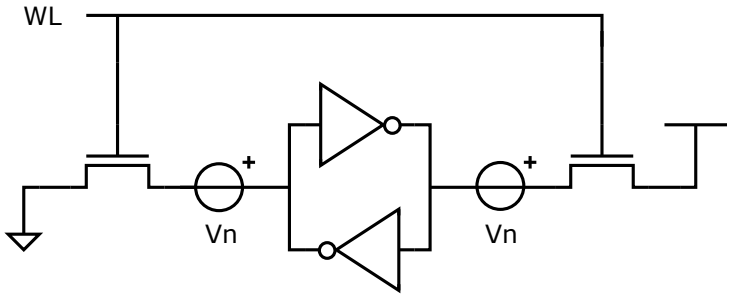

Figure 2.6. Schematic employed to test the write margin of the 6T cell.

2.4 Memory cell layout

Once the cell is sized, the step to perform is the realization of the layout. It consists in drawing planar geometric shapes that represent the various material layers of which the circuit is physically composed of.
This part is very important for this work, since it allows to perform simulations closer to the real cases. In fact, knowing how a circuit is physically made is helpful in order to know its parasitics element. A parasitic element

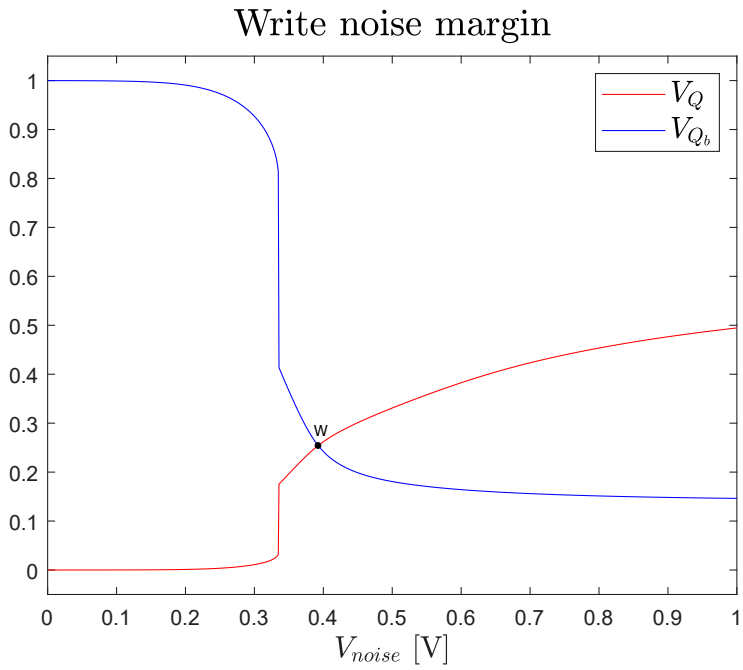

Figure 2.7. Internal nodes voltages as functions of the noise voltage during the write operation.

is an undesired electrical element possessed by a circuit, which contributes to the functioning of such circuit. In this case the parasitics which are taken into account are resistances and capacitances.

2.4.1 Description of the cell

Fig. x shows the layout of the SRAM standard 6t cell. It is possible to notice two main areas. The top one (green) is the *nwell* and the bottom one (orange) is the *pwell*. The PMOS and NMOS devices are drawn respectively in the nwell and pwell regions, and their active areas are clearly distinguishable in the drawing, having opposite color with respect to the well in which they are placed.
It is also possible to notice some vertical and horizontal bars drawn in metal1 (blue) and metal2 (violet). These metal interconnections are used to carry the signals through the whole circuit. The top horizontal one and the two outermost vertical ones are for the two power supplies: *vdd* and *gnd* respectively. While the other three bars are dedicated to *BL*, *BL_B*

and *WL* (respectively the two innermost vertical ones and the bottom horizontal one). In this layout, metal interconnections are drawn in a way such that adjacent cells are able to share them, implying area reduction.
Finally, the red geometries represent the polysilicon and are connected to the gate of all the devices which are present in the layout.

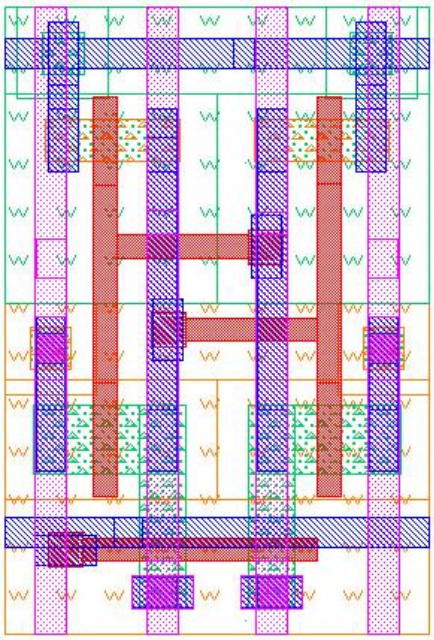

Figure 2.8. Layout of the 6T memory cell.

2.4.2 DRC and LVS

After that the layout is drawn, two essential steps have to be performed in order to be sure that the drawing is done correctly.
The first step is called **Design Rule Check (DRC)**. It consists in checking that the layout respects all the design rules imposed by the technology in use. Design rules are geometric constraints that must be respected in order to have a functioning and reliable design [8]. If the layout passes the DRC analysis, it is said to be "DRC clean". This fact does not ensure that the layout corresponds to the schematic of the designed circuit. To this aim, the second step have to be performed. It is called **Layout Versus Schematic (LVS)** [9]. If the layout passes the LVS analysis, it is said to

be "LVS clean".
At this point the design of the layout is complete.

In Table 2.1 is reported the final area of the 6T cells, resulting from the layout.

	6T Cell
Height (µm)	1.37
Width (µm)	0.93
Area (μm^2)	1.27

Table 2.1. Size of the 6T cell.

2.4.3 PEX

Once the layout is finished and correct, it must undergo a third fundamental step: the Parasitic Extraction (PEX). It consists in extracting from the layout the parasitic elements present in the circuit and creating a new schematic that takes them into account.
Once this step is done, the new schematic can be instantiated in the test benches to provide more realistic simulations.

2.5 Peripheral circuits characterization

In order to characterize all the implemented elements, a test bench schematic is drawn. As a first step, a monodimensional array of 6t cells is created. The number of the instantiated cells is arbitrarily set at the beginning. For this simulation (and for the ones discussed in the later sections) it is chosen to be 1024, which is a realistic number of cells for an SRAM array (e.g. the "1k x 8 SRAM" chips).
Each wordline is driven by a constant voltage generator: one of them is set to 1 V in order to select a word, while the others are set to 0 V to represent the unselected words. This is done because in standard SRAMs only one word at a time is selected for both the read and the write operations.
The peripheral circuits are connected to the memory cells through the bit-line couple.
All the simulations described in this section are transient analyses, where

the simulation parameters are the W of the transistors which make up each design.

2.5.1 Precharge circuit

The precharge circuit is the first peripheral circuit to be characterized.
The aim of the test is to find a W such that the circuit is able to precharge the bitline couple to V_{DD}. The precharge operation starts on the falling edge of the PRE# signal, when the PMOS transistors (whose gates receive a '0') connect the bitlines to V_{DD}. So, a pulsed voltage generator is used to drive the circuit, in order to simulate a transition of such signal.
The precharge time t_{PRE} is set to the value of 10 ns, which corresponds to one fifth of the total period T (50 ns).
For the purpose of the simulation, all the other signals are left to their default values, since during the precharge phase, none of the other components is meant to be working.

The value of Wp that allows the correct precharge of the bitlines is 1 µm.
The two circuital components that are not characterized yet (write driver and sense amplifier) contribute to the total capacitance of both BL and BL_B with a non-null component, even though they are not enabled during the precharge operation. For this reason, this value is considered as a first approximation, to be verified after the sizing of the devices in the schematics of the write driver and of the sense amplifier.

2.5.2 Write driver

The write driver must be able to discharge one of the to bitlines to gnd, depending on the value to be written into the memory cell. If the input data is '0', the node Q has to store a '0', so BL has to be discharged to gnd. While, in the opposite case ('1' has to be written into the memory cell), is BL_B that has to be discharged to gnd; in order to store a '0' on Q_B, and so a '1' on Q.
As for the precharge circuit, also the write driver has to be driven by a pulsed voltage generator. Such generator is responsible for the generation of the enable signal: WR_EN. This signal is driven to have a rising edge 2 ns after the one of PRE#. Its duration, t_{WR} is equal to 20 ns.

By activating the WL signal on the rising edge of PRE#, the write operation is correctly performed on the selected memory cell.
In order to characterize the cell it is necessary to test both the aforementioned cases.

The employed write driver have two couples of symmetric NMOS transistors(M1,M2 and M3,M4 referring to Figure 1.12). Different tests are performed; in which W values are varied differently. For instance, W of the M1 and M2 is set and W of M3 and M4 is swept, and viceversa. The best result, though, is obtained for the case in which the four devices in this design share the same W. The value of Wn_wr that allows the correct write operation (for both '0' and '1') is 150 nm.

2.5.3 Sense amplifier

The chosen topology for the sense amplifier generates its output on the bitline couple itself.
The transistors that make up this circuit cannot present all the same size. There are two inverters whose output is connected to the input of the other one, exactly in the same way as in the standard 6t SRAM cell. As reported in subsection 1.2.2 the sizes of the pull-up and pull-down transistors are chosen taking into account the currents that flow in them.
The circuit needs to be driven by two pulsed generators, responsible for the signals SE_EN and SE_EN_B, which, as the names suggest, are one the complementary version of the other. These two signals drive respectively M2 and M5, which are the devices that connect the amplifier to V_{DD} and gnd, enabling the bitline couple sensing.

The best W resulting after the parametric analyses are the following ones:

- $Wn\_sense = 5\,\mu m$ - W of the two pull-down NMOS.
- $Wn\_access = 4.5\,\mu m$ - W of the NMOS that connects the amplifier to gnd.
- $Wp\_sense = 1\,\mu m$ - W of the two pull-up PMOS.
- $Wp\_access = 4.5\,\mu m$ - W of the PMOS that connects the amplifier to V_{DD}.

2.6 Simulations

Once that all the circuital components are sized, and the layout undergoes the parasitic extraction, the simulations are performed. They are four: the writing of a '0' on a cell which is storing a '1'; the writing of a '1' on a cell which is storing a '0'; the reading of a '0'; the reading of a '1'.
In the following figures are reported the most significant signals involved in each operation.

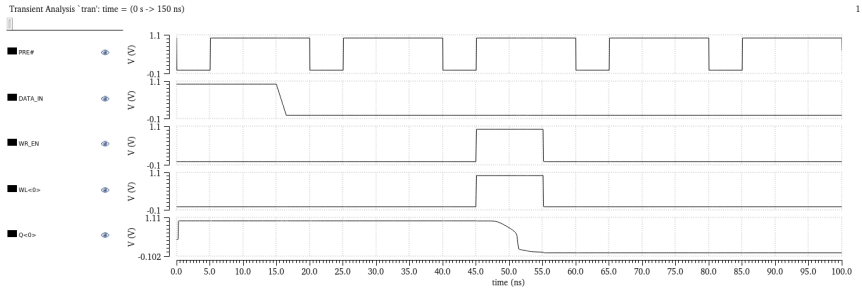

Figure 2.9. Write operation of a '0' on the Q node of a standard 6T cell.

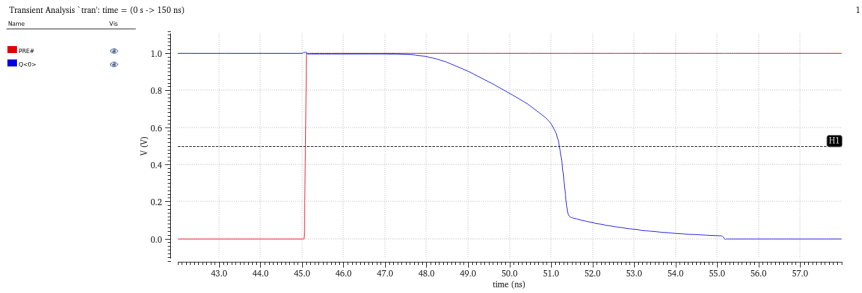

Figure 2.10. Write operation of a '0' on the Q node of a standard 6T cell: detail around the rising edge of the PRE# signal.

For what concerns the write operation, it is important to monitor the voltage on the node Q of the cell in which the operation is performed (or similarly the Q_B node). For this purpose Figure 2.9 (writing of a '0') and Figure 2.11 (writing of a '1') show the following signals:

- PRE#: which toggles periodically; the write operation can take place only during the evaluation phase (when PRE# is '1').

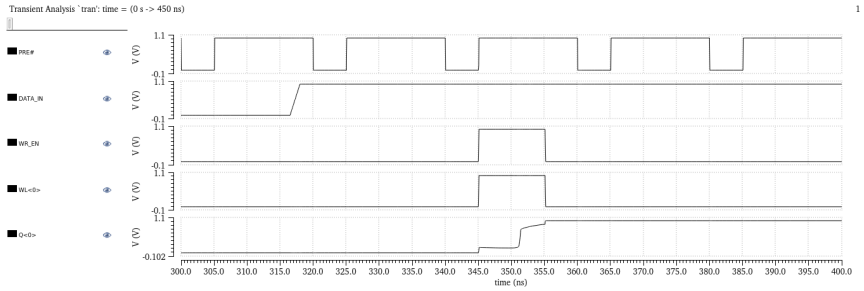

Figure 2.11. Write operation of a '1' on the Q node of a standard 6T cell.

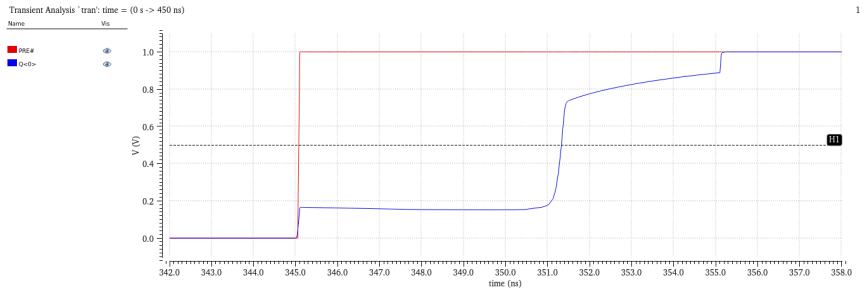

Figure 2.12. Write operation of a '1' on the Q node of a standard 6T cell: detail around the rising edge of the PRE# signal.

- DATA_IN: the value which has to be written in the selected cell.
- WR_EN: has to go high when PRE# is high to start the writing.
- WL<0>: has to go high along with WR_EN to select the cell in which DATA_IN has to be written (e.g. the cell with index 0).
- Q<0>: the node whose value changes after the write operation (e.g. the cell with index 0).

Figure 2.10 and Figure 2.12 show a detail on the rising edge of the PRE# signal, in the period in which the writing takes place.
It is possible to compute the delay that undergoes from the rising edge of the PRE# and the flipping of Q. This is done by taking into account the time interval between the two signals when they reach the 50% of their swing (i.e. 500 mV). The delays for the write operations are reported below in Table 2.2 along the ones for the read operations.

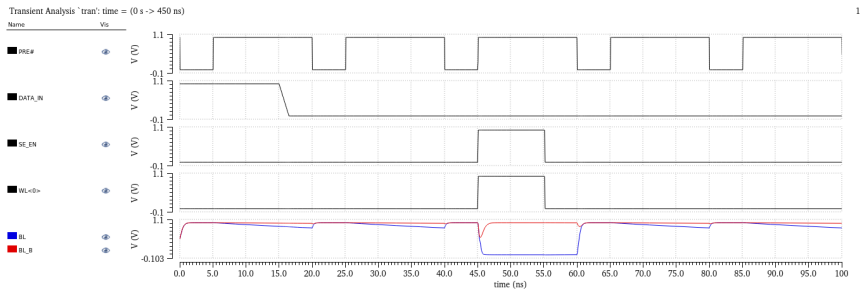

Figure 2.13. Read operation of a '0' on the BL/BL_B signals in a 6T SRAM array.

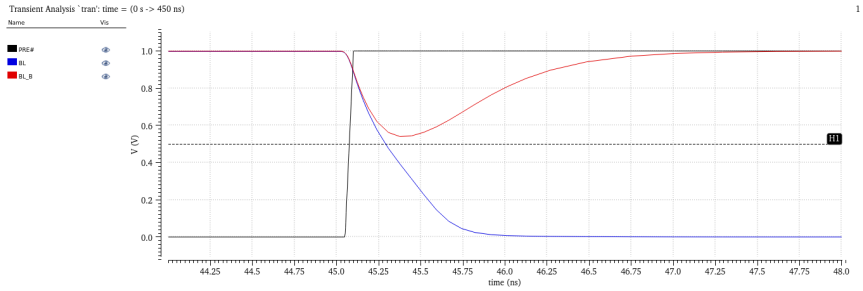

Figure 2.14. Read operation of a '0' on the BL/BL_B signals in a 6T SRAM array: detail around the rising edge of the PRE# signal.

Similarly, for what concerns the read operation, it is important to monitor the voltage on the bitline couple. For this purpose Figure 2.13 (reading of a '0') and Figure 2.15 (reading of a '1') show the following signals:

- PRE#: which toggles periodically; the read operation can take place only during the evaluation phase (when PRE# is '1').
- DATA_IN: the value previously written in the cell. For the purpose of checking if the cell content is read correctly, also looking at the Q<0> signal itself is useful.
- SE_EN: has to go high when PRE# is high to start the sensing.
- WL<0>: has to go high along with SE_EN to select the cell whose value has to be read on the bitline couple (e.g. the cell with index 0).
- BL/BL_B: the value of the two signals varies depending on the data in the cell that is being read (e.g. the cell with index 0).

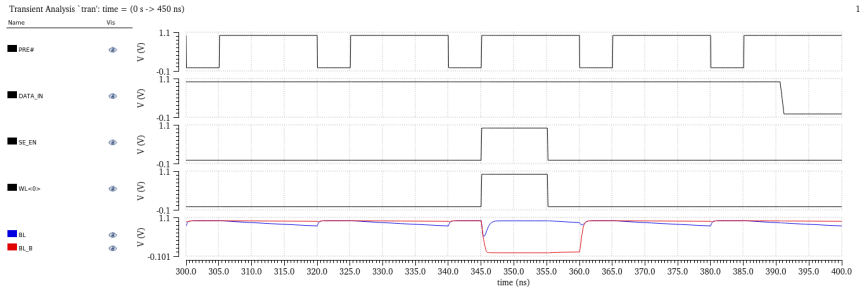

Figure 2.15. Read operation of a '1' on the BL/BL_B signals in a 6T SRAM array.

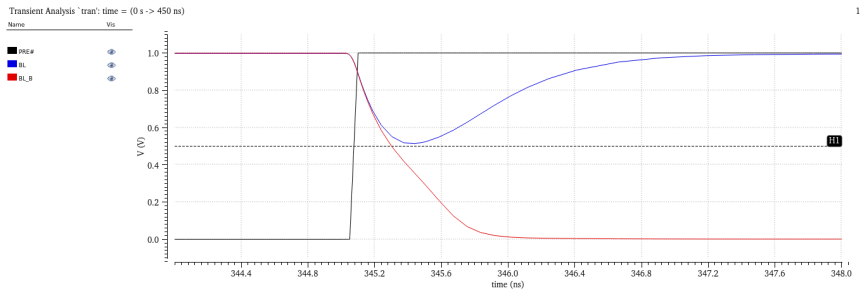

Figure 2.16. Read operation of a '1' on the BL/BL_B signals in a 6T SRAM array: detail around the rising edge of the PRE# signal.

Figure 2.14 and Figure 2.16 show a detail on the rising edge of the PRE# signal, in the period in which the sensing takes place.
It is possible to compute the delay that undergoes from the rising edge of the PRE# and the sensing on BL and BL_B. This is done by taking into account the time interval between the two signals when they reach the 50% of their swing (i.e. 500 mV).

Table 2.2 shows the results of the delay measurement in the four cases described above.

Another important metric to take into account when analysing an electronic circuit is the consumption. In this work, to evaluate the consumption of the implemented cells, energy is preferred rather then power.

	6T Cell
DELAY WR0 (ns)	5.56
DELAY WR1 (ns)	6.25
DELAY SE0 (ns)	0.21
DELAY SE1 (ns)	0.22

Table 2.2. Delays for the 6T cell.

	6T Cell
ENERGY WR0 (fJ)	107.90
ENERGY WR1 (fJ)	24.35
ENERGY SE0 (fJ)	15.26
ENERGY SE1 (fJ)	16.59

Table 2.3. Energy consumption for the 6T cell.

	6T Cell
POWER WR0 (µW)	19.41
POWER WR1 (µW)	3.89
POWER SE0 (µW)	70.65
POWER SE1 (µW)	73.41

Table 2.4. Power consumption for the 6T cell.

Chapter 3

The Logic Gates

3.1 The dynamic logic

The core idea behind this work is that logic operations can be executed inside the memory cell itself. So, with this in mind, there is one important choice to make: how to implement the functions at transistor-level. The logic functions chosen for the implementation are the AND, the OR and the XOR. This choice can be explained by the fact that these logic operations are among the most used ones. It is also quite immediate to derive their "complementary" versions (i.e. NAND, NOR and XNOR) starting by them.
The area occupation is a fundamental metrics to evaluate the feasibility of the implementation, so the choice made is to employ the so called *dynamic logic*, which needs a lesser number of transistors. This logic family is also faster w.r.t. the static one due to the smaller capacitive loads which are charged and discharged when the operations are performed [7].
The dynamic logic family operates on two phases, regulated by a clock signal: the precharge one (clock = '0') and the evaluation one (clock = '1'). During the precharge phase the output becomes '1', and during the evaluation phase, it may change its value to '0' if a path towards gnd is created.
For this purpose, a dynamic logic gate is made of a pMOS, responsible for the precharge of the output, and some nMOS which implement the logic function itself. Another nMOS is employed in order to prevent the current flow from V_{DD} to gnd during the precharge phase.

As for the static CMOS logic, the dynamic logic is more suitable to implement inverting functions. Since the aim of this work is to implement non-inverting functions, one solution is the one of exploiting the *De Morgan's laws*.

3.2 De Morgan's laws

De Morgan's laws (or theorems) [10] are two logical equivalences named after the British mathematician Augustus De Morgan. They can be expressed in English as follows:

1. The negation of a disjunction is the conjunction of the negations.
2. The negation of a conjunction is the disjunction of the negations.

Knowing that in Boolean algebra the disjunction operator is the OR and the conjunction operator is the AND, these theorems can be easily applied to logic gates. As a matter of fact, the two equivalences become:

$$\overline{A \; or \; B} = \overline{A} \; and \; \overline{B} \tag{3.1}$$

$$\overline{A \; and \; B} = \overline{A} \; or \; \overline{B} \tag{3.2}$$

Looking at them is it clear the relationship that exists between the OR and the NAND logic operations; and the one between the AND and the NOR ones. An AND gate can be realized negating the inputs of a NOR gate and, viceversa, an OR gate can be realized negating the inputs of a NAND one.
Since in an SRAM cell, it is possible to access not only to the Q node, but also to its complementary one (Q_B), it is easy to implement an AND gate by exploiting (3.1) and an OR gate by exploiting (3.2).

3.3 Gates design

Dynamic logic gates can be easily sized. The only pMOS in each cell is left with the minimum W possible (i.e. 90 nm). This is done because the duration of the precharge phase is determined by the clock of the gate (i.e. the PRE# signal in Figure 4.2, Figure 4.3 and Figure 4.4).

Concerning the nMOS transistors, the size of their W has to be determined considering how many of them are in series, i.e. how many of them can be active at the same time. The resistance of an nMOS device with $W = k * W_{min}$ is R/k (where R is the resistance of the nMOS with $W = W_{min}$.
As a consequence, the more they are, the larger they have to be, in order to let the same amount of current to flow in them. So, theoretically, in a series of two nMOS, their W has to be equal to $90\,\text{nm} * 2 = 180\,\text{nm}$, while in a series of three, $90\,\text{nm} * 3 = 270\,\text{nm}$.

3.3.1 Dynamic AND gate

To design an AND gate with dynamic logic, and exploiting Q_B, it is sufficient to implement the pull-down network of a static CMOS NOR gate, using Q_B and an external input, called IN_B. By substituting A and B in (3.1) with BL_B and IN_B, and rearranging the equation, one ends up with a gate which implements the function **Q AND IN**.

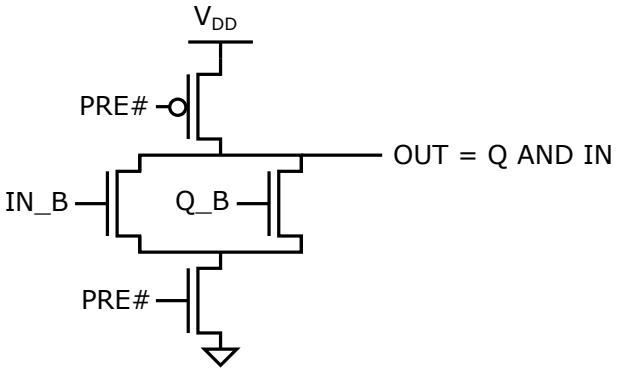

Figure 3.1. Dynamic NOR gate used to implement the logic function Q AND IN.

3.3.2 Dynamic OR gate

Similarly to what is said for the AND gate, the design of the OR gate is done substituting A and B in (3.2) with BL_B and IN_B, and rearranging the equation, obtaining a gate which implements the function **Q OR IN**.

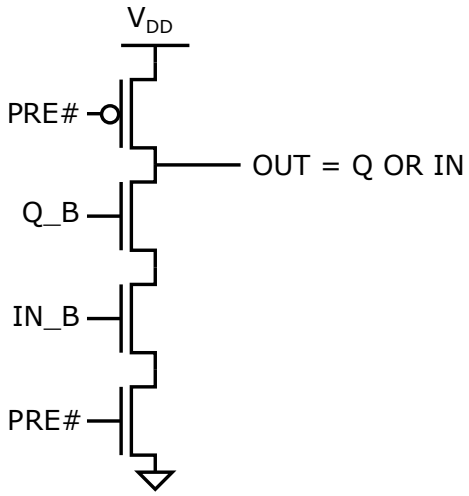

Figure 3.2. Dynamic NAND gate used to implement the logic function Q OR IN.

3.3.3 Dynamic XOR gate

For what concerns the XOR logic function, both the negated and non-negated version of the input signals are needed, so an additional signal IN is needed (along with its complementary IN_B as for the two previous logic functions). The static CMOS version of this gate is already a non-inverting one, so it is sufficient to adopt its pull-down network to implement the function **Q XOR IN**.

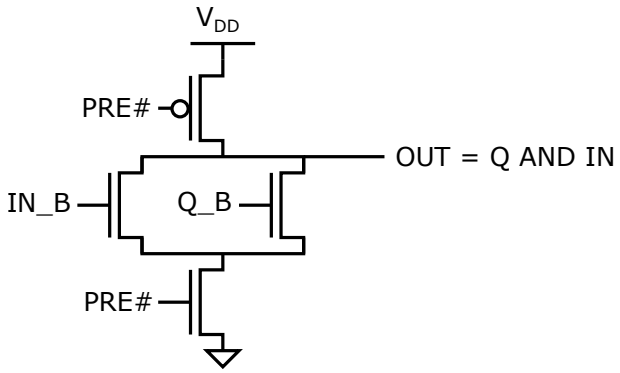

Figure 3.3. Dynamic XOR gate used to implement the logic function Q XOR IN.

Chapter 4

Logic-in-Memory Cells

4.1 Workflow

As it is described in chapter 2, the SRAM array composed of standard 6T cell is complete.
The next step in this thesis work is the design of LiM memory cells.
The workflow to follow for each new cell is the same one adopted for the 6T cells. The first step is the design of the LiM cell, starting from the implemented standard 6T. Once that the schematic is designed, the added transistors have to be sized. To do so, it is necessary to perform parametric analyses. The size of the peripheral circuitry is kept the same.
The simulations to be performed concern the write operation, the read operation, the data retaining and the logic operations. The employed test benches include a precharge circuit, a write driver and two sense amplifiers: one for the BL/BL_B sensing, as in the standard case, and one for the logic output sensing.
After that all the involved devices are sized in a way that grants the correct functioning of the cell, the layout has to be drawn.
As final steps, after that the layout correctly passes the DRC and LVS analyses, the parasitic elements have to be extracted, and the simulation have to be performed with the extracted view.

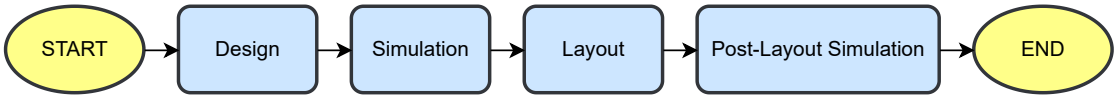

Figure 4.1. Workflow followed for each LiM cell.

4.2 Schematics of the LiM cells

In order to put the LiM paradigm into practice, the inputs of a dynamic logic gate are driven by the node Q_B of a 6T SRAM cell and an external signal (referred to as IN_B, before). By simply adopting this approach, three starting LiM cells are designed: one which implements the AND operation, one which implements the OR operation and one which implements the XOR operation.

4.2.1 The LiM-AND cell

The first LiM cell to be designed is the one implementing the logic AND function. As it can be seen from Figure 4.2, the circuit is nothing but the combination of a standard 6T memory cell and a dynamic NOR gate. As said in chapter 3, driving a NOR gate with the complemented version of two signals, means to realize an AND gate.
The transistors which make up the logic gate inside the LiM cell, have to be sized. To do so, a first parametric analysis is run, where the parameter to be swept is the width of the nMOS devices. The width of the pMOS, instead, is set to the minimum (i.e. 90 nm) since there are no strict constraints on the precharge operation in terms of speed. From the parametric transient analysis, results that also the nMOS can be left with the minimum width possible.

4.2.2 The LiM-OR cell

The second LiM cell to be designed can be considered as the dual of the previous one. In fact, the circuit, shown in Figure 4.3, is a combination of the standard 6T memory cell and a dynamic NAND gate. Recalling also here what said in chapter 3, driving a NAND gate with the complemented version of two signals, means to realize an OR gate.
Concerning the size of the precahrge pMOS, the consideration that can be

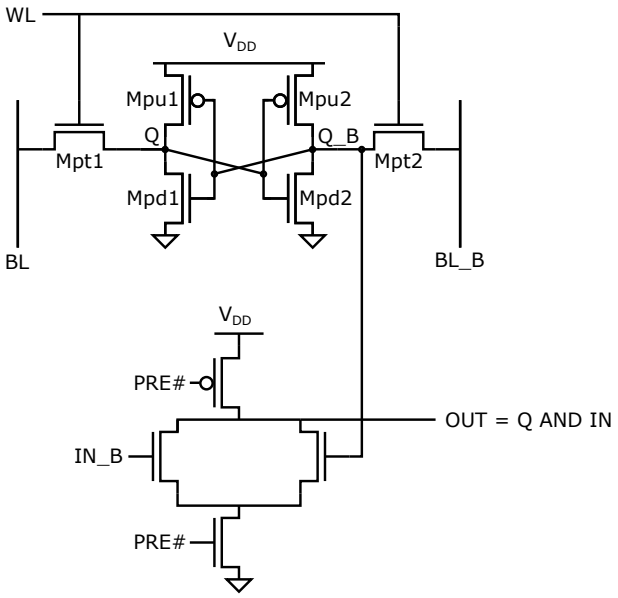

Figure 4.2. Schematic of the LiM-AND cell.

made is the same as the one done for the previous cell: its width can be set at 90 nm. Instead, the parametric transient analysis give a different result about the size of the pull-down devices: the size that allow the correct functioning of the circuit both as a memory cell and as a logic combinatorial circuit is 180 nm.

4.2.3 The LiM-XOR cell

Concerning the XOR function, the cell which implements it is designed in a similar way w.r.t. the previous two. In order to carry out a XOR operation, both the complemented and the standard version of the inputs are needed. To fulfill this requirement, the schematic shown in Figure 4.4 is designed.
Transistor sizes are the same ones as for the LiM-OR cell. The number of nMOS devices in series is the same one as in the previous cell, so this implies that the same current flows in a branch of the circuit. This is always true for the dynamic XOR gate, since current cannot flow in the two branches at the same time, due to the signals that drive the gates of the devices.

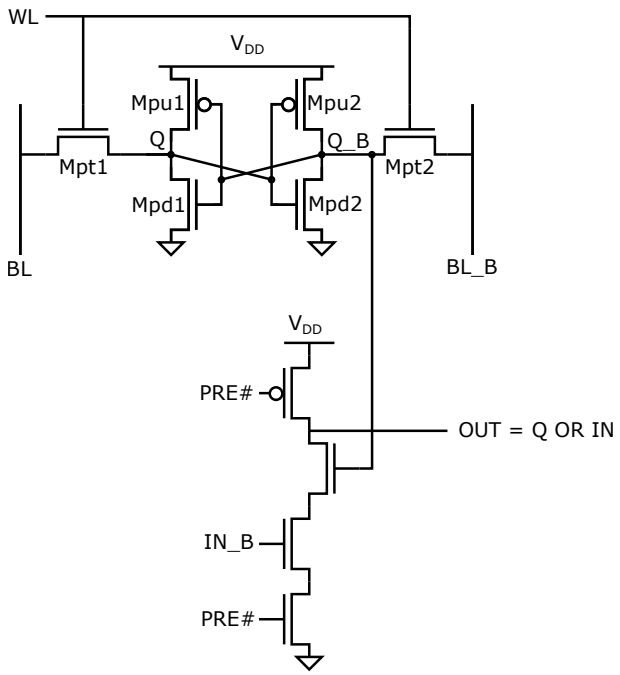

Figure 4.3. Schematic of the LiM-OR cell.

4.2.4 The Write-Back XOR cell

After the previously analyzed LiM-XOR cell, another cell which carries out the same operation is designed. This is done because up to this point, the outputs of all the LiM elements (i.e. the outputs of the logic gates) are sent out of the cells.
From the perspective of saving power and time, it would be useful to have a memory array from which the elaborated data is sent out as little as possible. A cell which is able to write its content inside a cell belonging to another word is a possible solution to accomplish this need. This feature is referred to as *"write-back"* [11].

The cell in Figure 4.5 exhibit an additional nMOS device, which acts as a pass transistor. In fact,the task of M13 is to pass the value of the signal IN_CELL into the node Q. The signal which drives the gate of M13, is CL (which stands for "Control Line").
In an array composed of XOR-WB cells, every input IN_CELL is connected to the output OUT_XOR of the cell sharing the same bitline but belonging to the previous word.

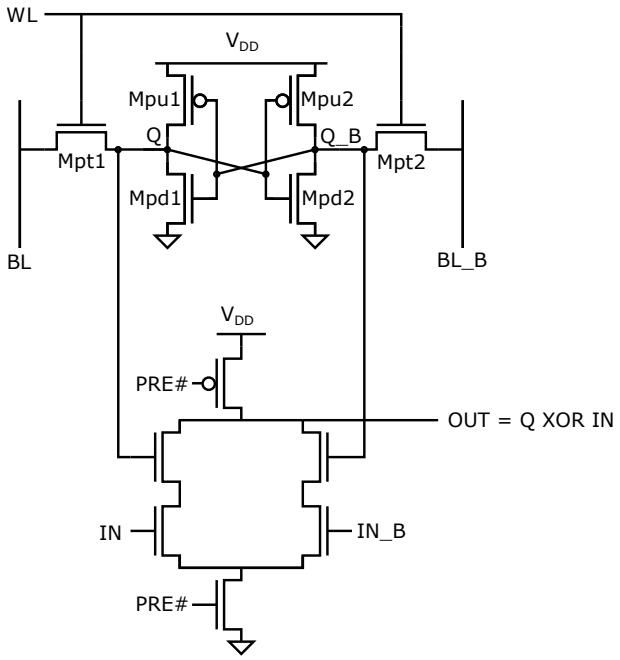

Figure 4.4. Schematic of the LiM-XOR cell.

If all the cells share the same CL, it is possible to write back a whole word into the successive one. This operation may be useful, for example, to apply a mask to a word saved in the memory, and to directly store the result in another memory location (in this case the successive one).
The sizes of the additional devices w.r.t. the standard 6T cell, are determined by performing parametric analyses, also in this case. For this cell, more than one analysis is needed, since both the combinatorial and the write-back operations need to be tested. The two are not totally unrelated, since the logic output of a cell is connected to the drain of the pass-transistor of the successive one in the same bitline. The minimum width of the pass-transistor that allows the correct writing of the external data on the node Q, is 500 nm. This value can be explained by the fact that this device has to "win" over the access pass-transistor that connects the node Q to the BL. To do so, it has to force a higher current, and as a consequence, it is possible to obtain

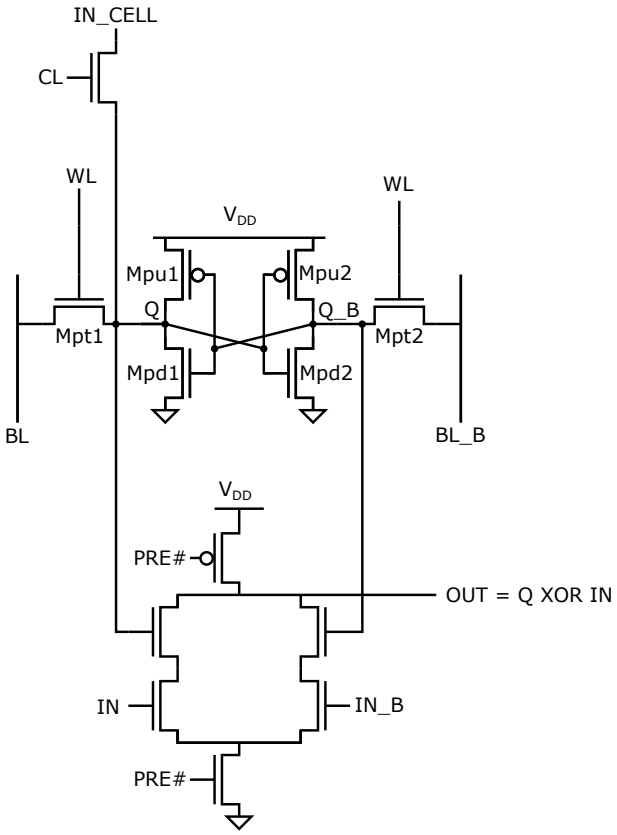

Figure 4.5. Schematic of the LiM-XOR-WB cell.

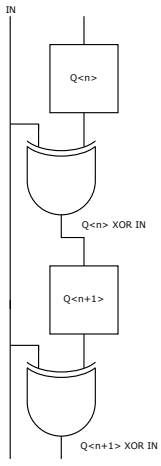

Figure 4.6. Block scheme which depicts the write-back operation.

4.3 Layouts of the LiM cells

The next step in the workflow (Figure 4.1) is the realization of the IC layouts. The importance of the layout is already discussed in chapter 2, and the same considerations made also hold for the circuits designed in this Chapter.
Each layout presented in this section is drawn by adopting the same procedure seen in chapter 2. Hence, DRC analysis, LVS analysis and PEX are performed.
Since the fact that the new layouts will occupy more are w.r.t. the 6T is obvious (due to the higher number of MOS transistors), these drawings are made trying to leave the area occupation as low as possible, while respecting the design rules.

4.3.1 The LiM-AND cell

For the LiM-AND cell two different layouts are drawn. The aim is to compare the results of the two in terms of performances, understanding and evaluating the impact of the layout.

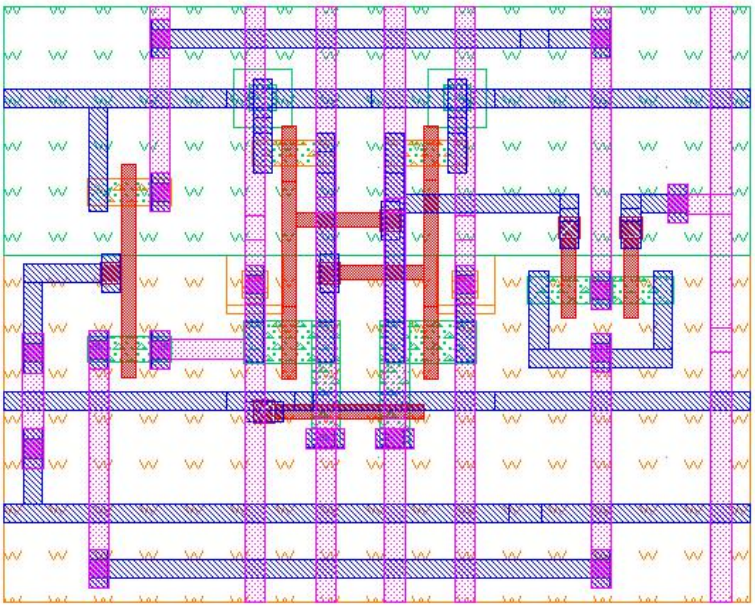

Figure 4.7. First layout of the LiM-AND cell: AND-v1.

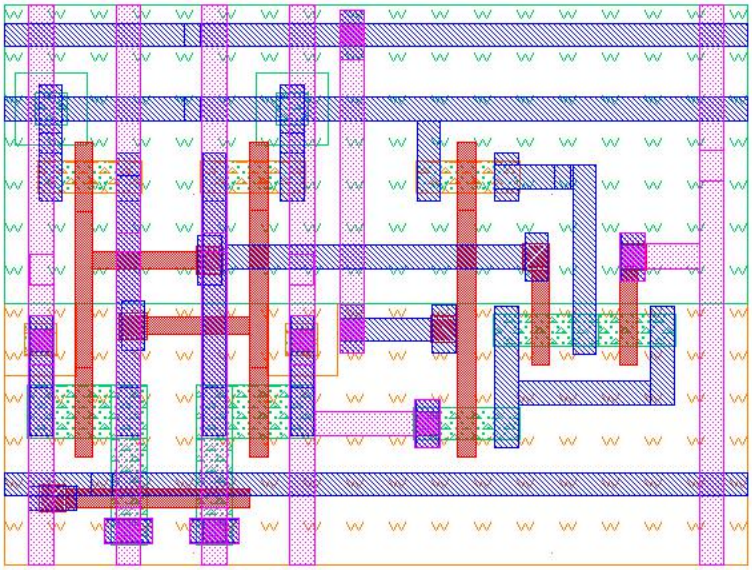

Figure 4.8. Second layout of the LiM-AND cell: AND-v2.

Figure 4.7 and Figure 4.8 show the two different versions, called "AND-v1" and "AND-v2". The first version presents a more symmetric layout, taking as reference the six transistors that make up the 6T cell. Instead, the second version exhibits all the transistors of the logic gate on the right of the ones of the memory, making the layout asymmetric. At this point it is only possible to make considerations about the size of the different layouts. After that post-layout simulations are run, it will be possible to compare the results in terms of delay and consumption.

	AND-v1	AND-v2
HEIGHT (µm)	2.06	1.56
WIDTH (µm)	2.59	2.07
AREA (μm^2)	5.33	3.23

Table 4.1. Sizes of the layouts AND-v1 and AND-v2 for the LiM-AND cell.

Table 4.1 collects the sizes of the AND-v1 and AND-v2 layouts. It is possible to notice that adopting the second version, the cell undergoes a reduction in both height and width, allowing to save 39.4 of the area occupation due to the cells.

4.3.2 The LiM-OR cell

The very same procedure is adopted for the LiM-OR cell.

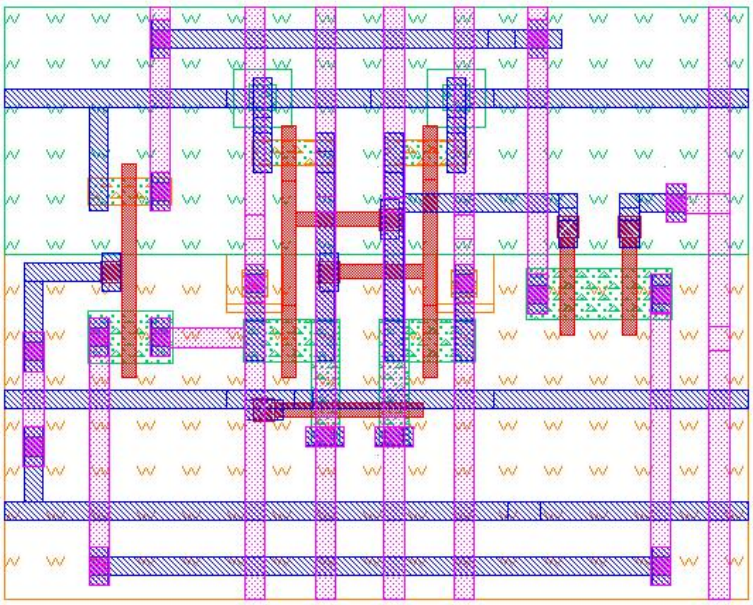

Figure 4.9. First layout of the LiM-OR cell: OR-v1..

Similarly, Figure 4.9 and Figure 4.10 show the two different layout versions, namely "OR-v1" and "OR-v2". The differences in the two are the same ones presented for the two previous layouts.

	OR-v1	OR-v2
HEIGHT (µm)	2.06	1.56
WIDTH (µm)	2.59	2.07
AREA (μm^2)	5.33	3.23

Table 4.2. Sizes of the layouts OR-v1 and OR-v2 for the LiM-OR cell.

In Table 4.3 are reported the sizes of the OR-v1 and OR-v2 layouts. It can be observed that the two maintain the exact same widths and heights.

4.3.3 The LiM-XOR cell

In this work, only one layout of the LiM-XOR cell is realized. This is because the cell is needed as a starting point in the design of the one

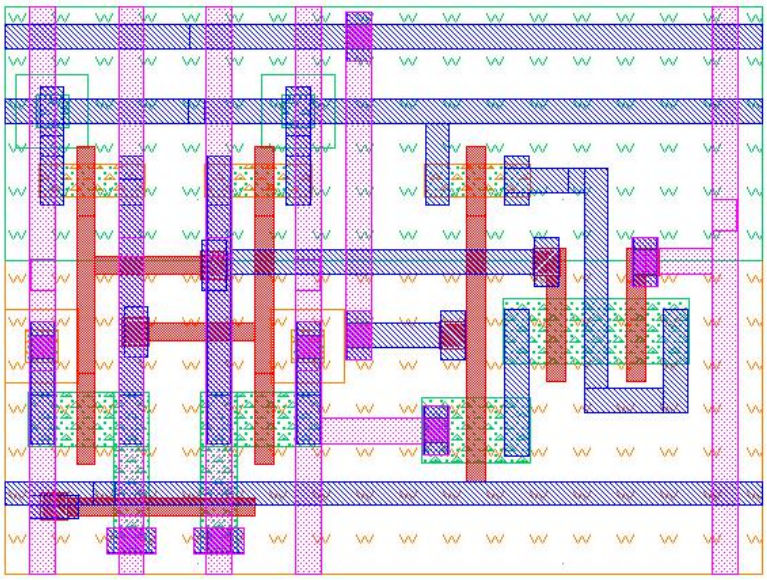

Figure 4.10. Second layout of the LiM-OR cell: OR-v1.

executing the write-back operation. It is shown in Figure 4.11.

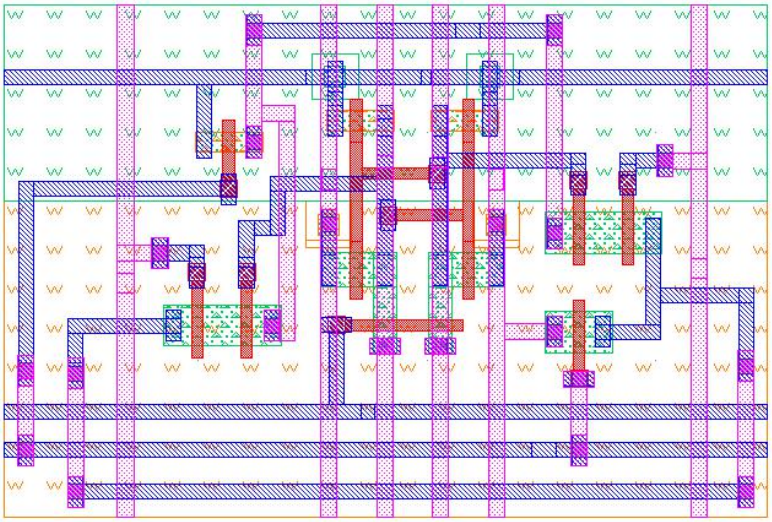

Figure 4.11. Layout of the LiM-XOR memory cell.

4.3.4 The Write-Back XOR cell

Also for the last LiM cell, two different IC layouts are realized. The two are presented in Figure 4.12 and Figure 4.13.

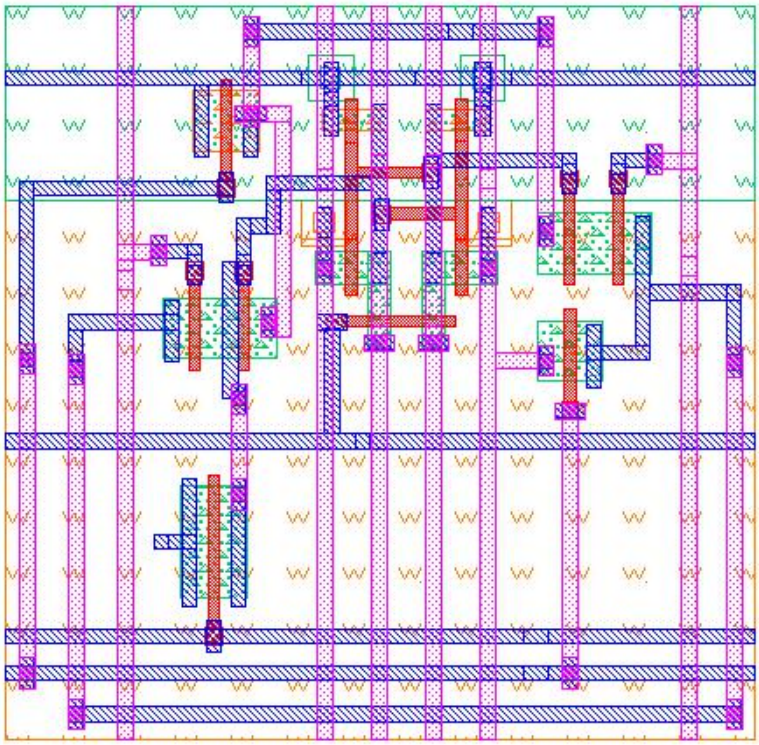

Figure 4.12. First layout of the LiM-XOR-WB cell: XOR-WB-v1.

	XOR	XOR-WB-v1	XOR-WB-v2
HEIGHT (µm)	2.24	3.25	2.43
WIDTH (µm)	3.35	3.35	3.86
AREA (μm^2)	7.47	10.85	9.37

Table 4.3. Sizes of the layout for the LiM-XOR cell and the layouts XOR-WB-v1 and XOR-WB-v2 for the LiM-XOR-WB cell.

4.3.5 Parasitic Extraction from layouts

After that IC layouts are realized it is possible to move forward with the parasitic extraction process. Parasitic contributions are taken into account

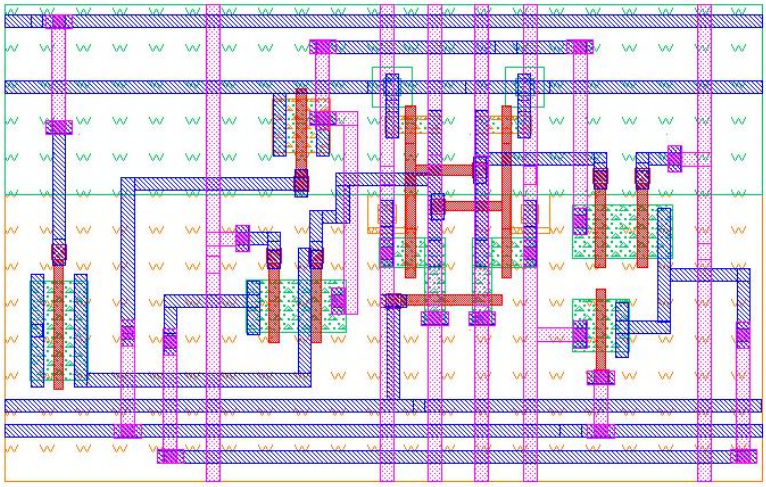

Figure 4.13. Second layout of the LiM-XOR-WB cell: XOR-WB-v2.

in the so called *Calibre View* in Cadence Virtuoso. It is necessary to substitute the previous *Schematic View* with the Calibre one in order to account for parasitic contributions in future analyses. This have to be done for each LiM cell in the corresponding test-bench schematic (the very same procedure done for the 6T cell in chapter 2).
In chapter 5 the results of post-layout simulations are discussed.

Chapter 5

Post-layout simulations

In this chapter, the results of post-layout simulations are presented and analyzed. When evaluating the feasibility of an integrated circuit, it is important to take into account the parasitic contributions, since ideal models are far from the physical implementation.
Each cell is tested in writing mode and reading mode. For both operations, the '0' case and the '1' case are differentiated and analyzed separately, since the LiM cells may behave asymmetrically w.r.t. the value which is written or read. The same consideration is made for the write-back operation performed by the LiM-XOR-WB cell.
The metrics taken into account in this part of the work are the cell access delay, the energy consumption and the power consumption (each of them evaluated for every operation). Delays are measured starting from the 50% of the rising edge of the precharge (PRE#) signal, up to the 50% of the transition of the signal of interest for each operation. Namely, if the operation under analysis is the memory writing, this signal is the voltage at the storage node Q of the cell. In the memory reading case it is represented by the BL voltage (or BL_B, depending on which of the two is discharged). Lastly, for the write-back, the signal of interest is the voltage at the node Q of the cell which is being written.
For what concerns the consumption, energy and power are computed namely with Equation 5.1 and Equation 5.2.

$$E = \int_{t_1}^{t_2} V(t) * I(t)\,\mathrm{d}t \tag{5.1}$$

$$P = \frac{1}{t_2 - t_1} \int_{t_1}^{t_2} V(t) * I(t)\, \mathrm{d}t \tag{5.2}$$

In the two equations, V(t) is the voltage of the signal involved in the operation and I(t) is the current drained from the power supply generator to gnd. Furthermore, t_2 - t_1 is the quantity identified before as the delay of the operation.
For every IC layout, four timing diagrams are reported in this document: one for the writing of a '0', one for the writing of a '1', one for the reading of a '0', and finally one for the reading of a '1'.

5.1 AND-v1

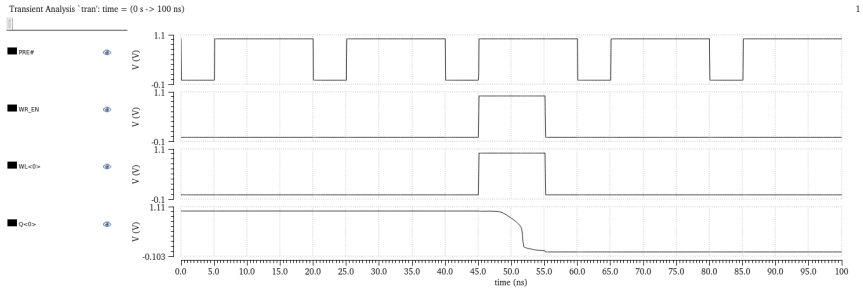

Figure 5.1. Write operation of a '0' on the Q node of a AND-v1 cell.

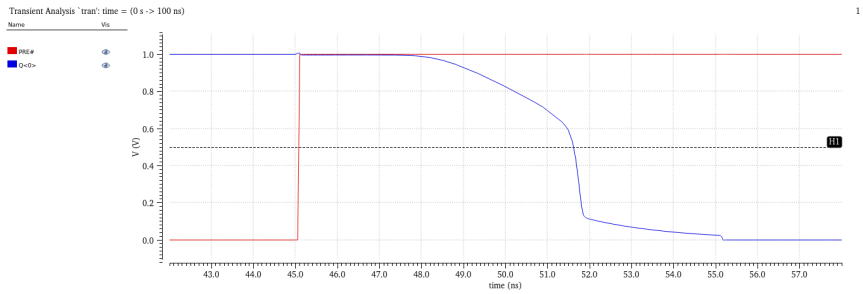

Figure 5.2. Write operation of a '0' on the Q node of a AND-v1 cell: detail around the rising edge of the PRE# signal.

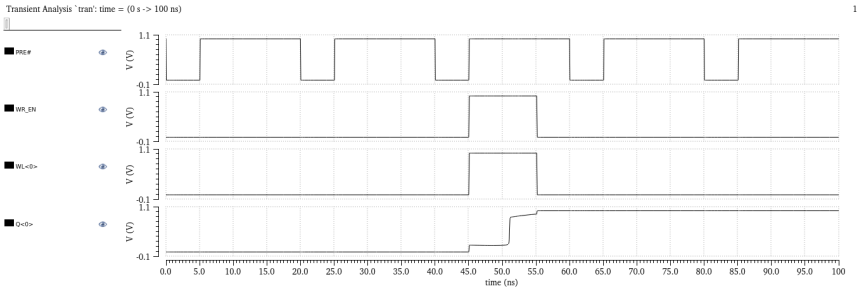

Figure 5.3. Write operation of a '1' on the Q node of a AND-v1 cell.

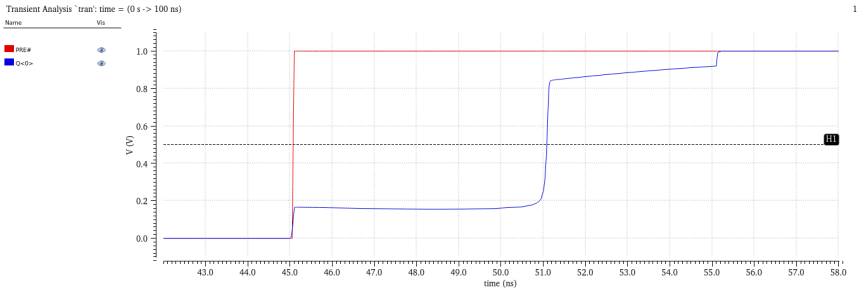

Figure 5.4. Write operation of a '1' on the Q node of a AND-v1 cell: detail around the rising edge of the PRE# signal.

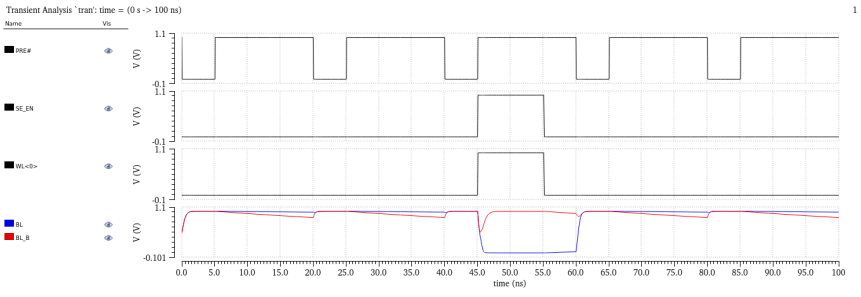

Figure 5.5. Read operation of a '0' on the BL/BL_B signals in a AND-v1 array.

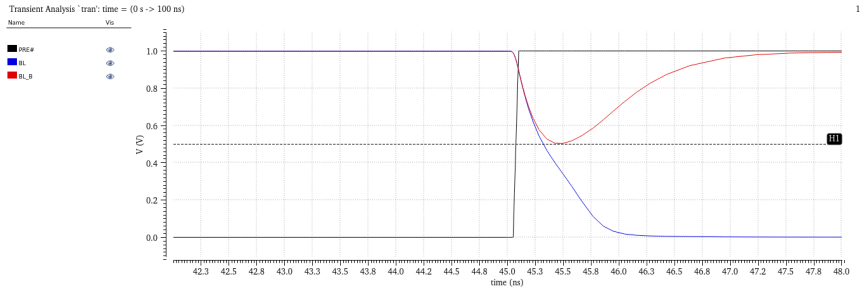

Figure 5.6. Read operation of a '0' on the BL/BL_B signals in a AND-v1 array: detail around the rising edge of the PRE# signal.

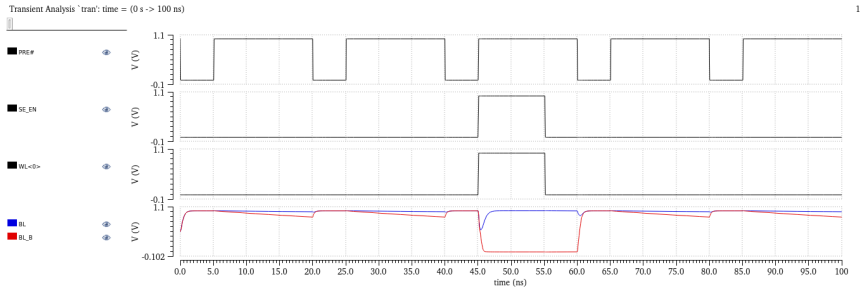

Figure 5.7. Read operation of a '0' on the BL/BL_B signals in a AND-v1 array.

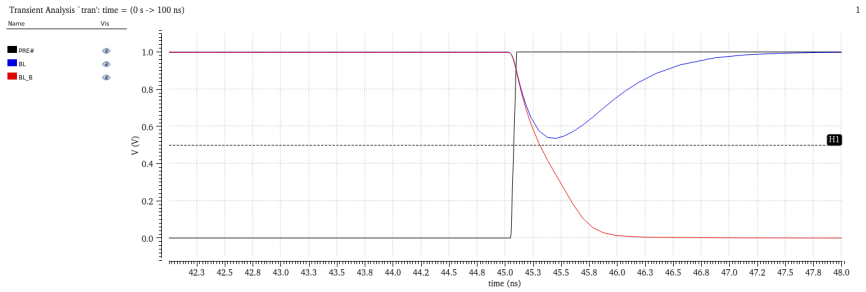

Figure 5.8. Read operation of a '1' on the BL/BL_B signals in a AND-v1 array: detail around the rising edge of the PRE# signal.

5.2 OR-v1

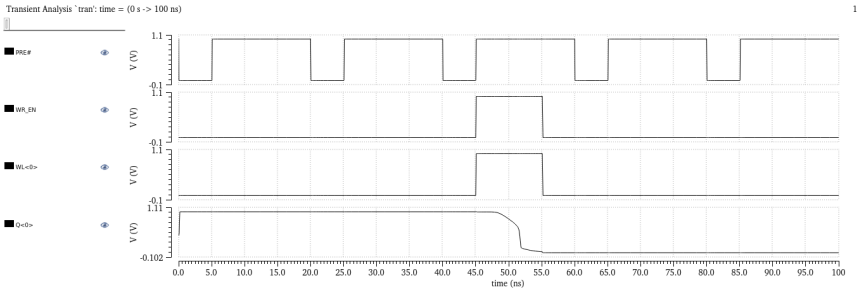

Figure 5.9. Write operation of a '0' on the Q node of a OR-v1 cell.

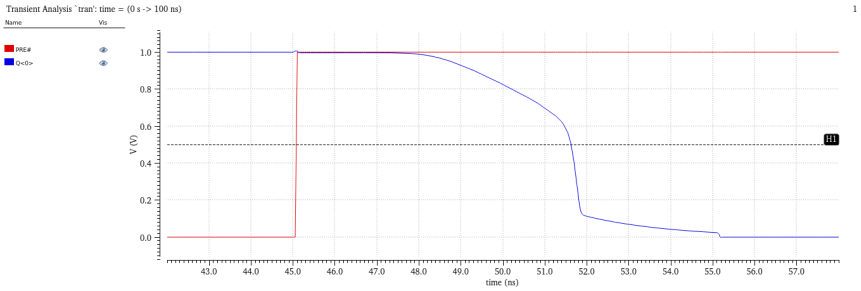

Figure 5.10. Write operation of a '0' on the Q node of a OR-v1 cell: detail around the rising edge of the PRE# signal.

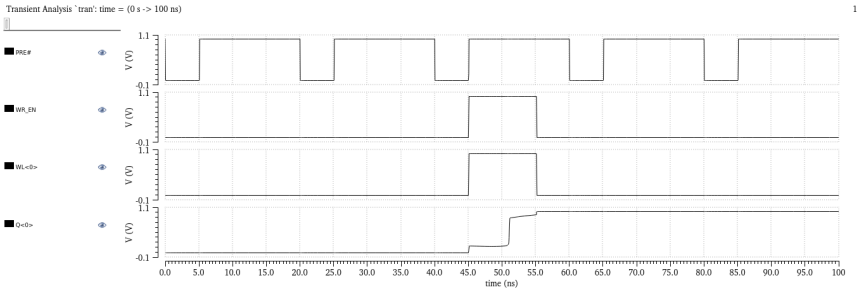

Figure 5.11. Write operation of a '1' on the Q node of a OR-v1 cell.

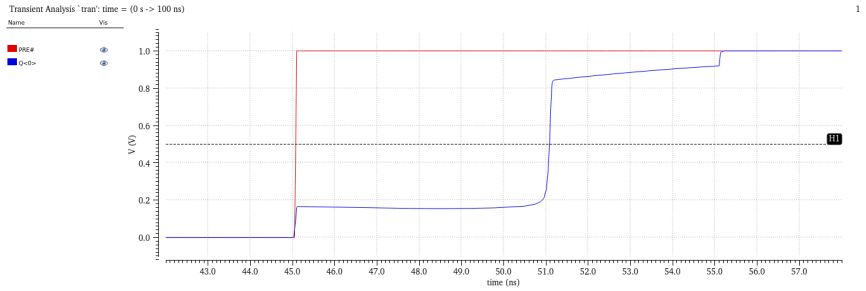

Figure 5.12. Write operation of a '1' on the Q node of a OR-v1 cell: detail around the rising edge of the PRE# signal.

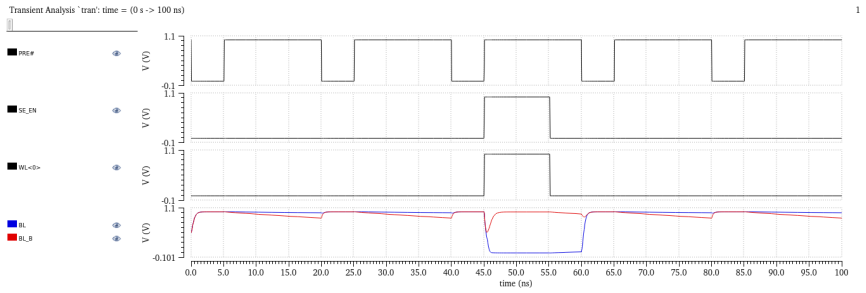

Figure 5.13. Read operation of a '0' on the BL/BL_B signals in a OR-v1 array.

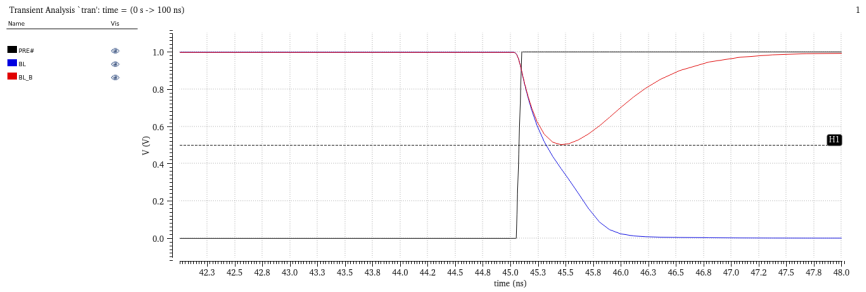

Figure 5.14. Read operation of a '0' on the BL/BL_B signals in a OR-v1 array: detail around the rising edge of the PRE# signal.

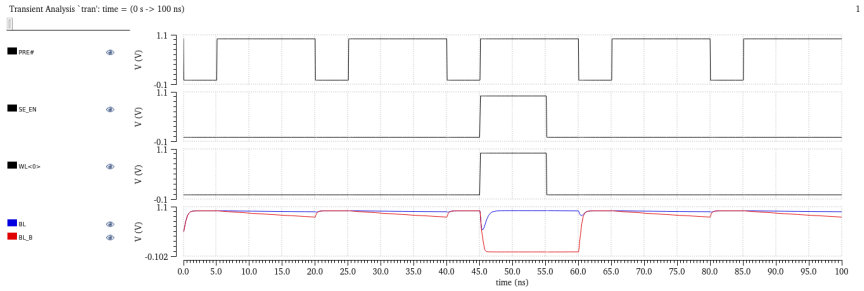

Figure 5.15. Read operation of a '1' on the BL/BL_B signals in a OR-v1 array.

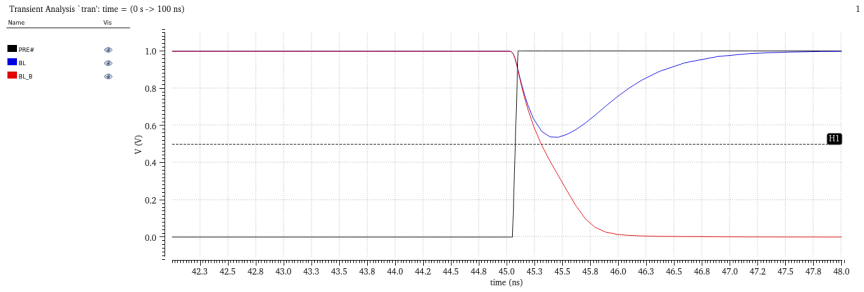

Figure 5.16. Read operation of a '1' on the BL/BL_B signals in a OR-v1 array: detail around the rising edge of the PRE# signal.

5.3 AND-v2

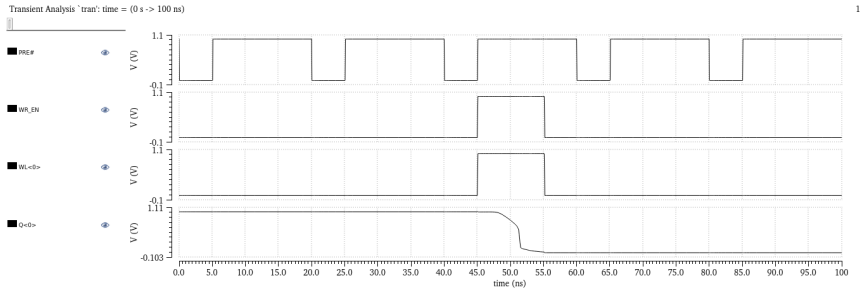

Figure 5.17. Write operation of a '0' on the Q node of a AND-v2 cell.

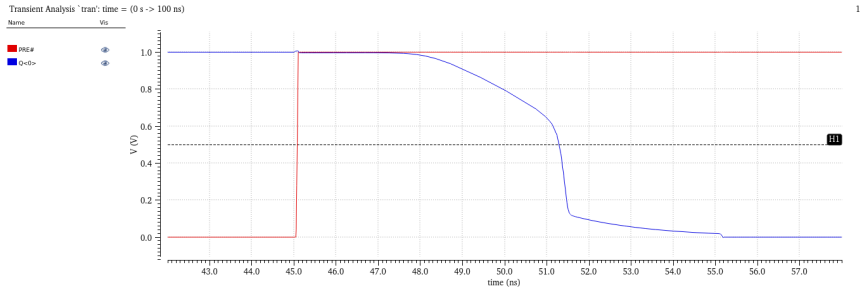

Figure 5.18. Write operation of a '0' on the Q node of a AND-v2 cell: detail around the rising edge of the PRE# signal.

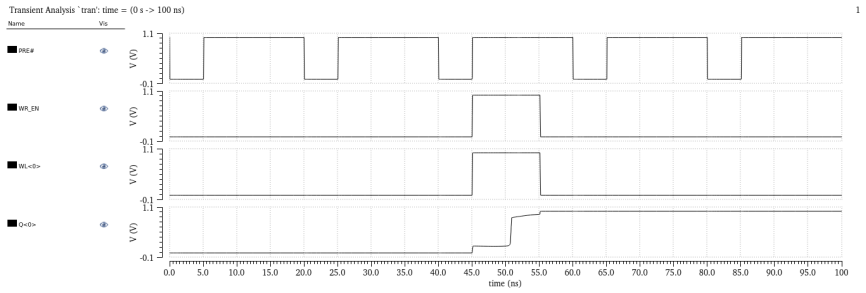

Figure 5.19. Write operation of a '1' on the Q node of a AND-v2 cell.

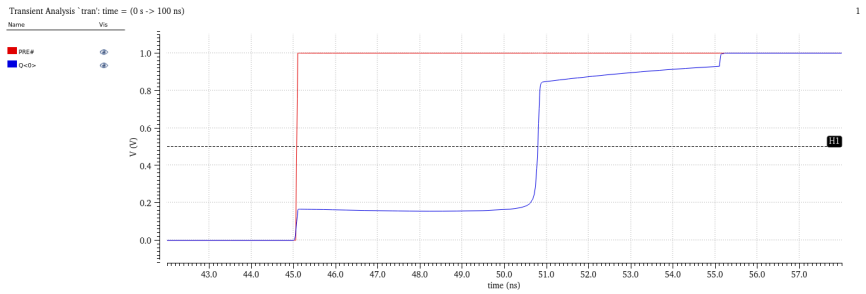

Figure 5.20. Write operation of a '1' on the Q node of a AND-v2 cell: detail around the rising edge of the PRE# signal.

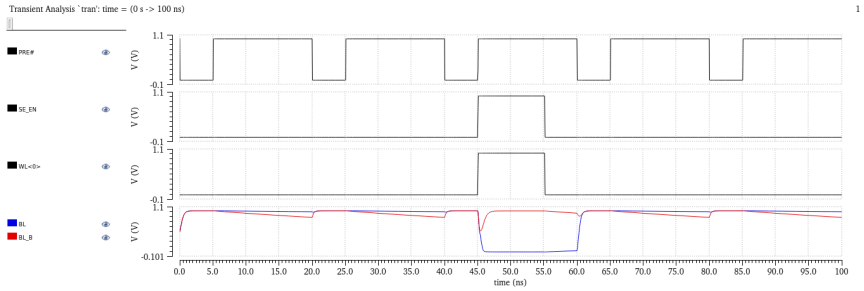

Figure 5.21. Read operation of a '0' on the BL/BL_B signals in a AND-v2 array.

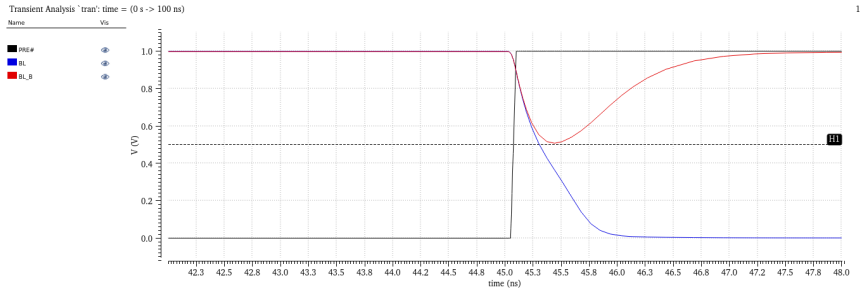

Figure 5.22. Read operation of a '0' on the BL/BL_B signals in a AND-v2 array: detail around the rising edge of the PRE# signal.

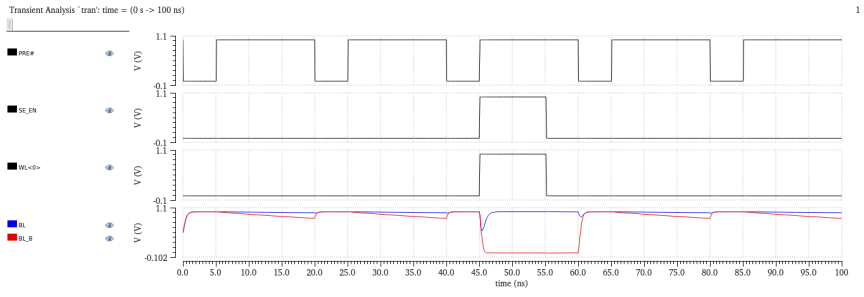

Figure 5.23. Read operation of a '0' on the BL/BL_B signals in a AND-v2 array.

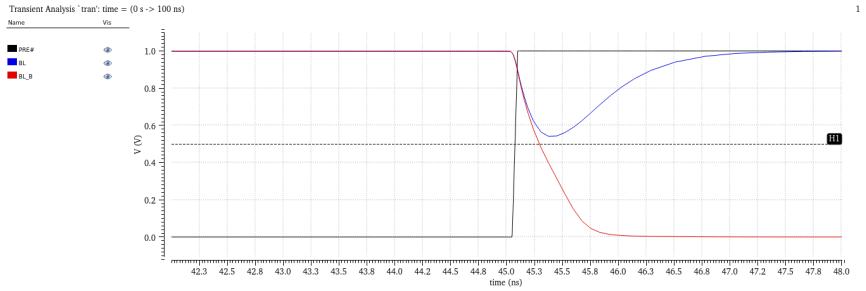

Figure 5.24. Read operation of a '1' on the BL/BL_B signals in a AND-v2 array: detail around the rising edge of the PRE# signal.

5.4 OR-v2

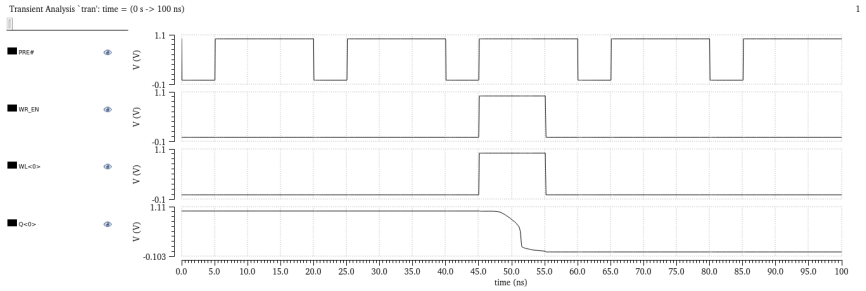

Figure 5.25. Write operation of a '0' on the Q node of a OR-v2 cell.

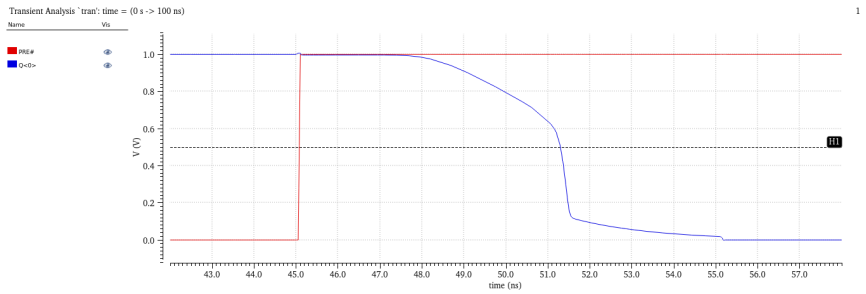

Figure 5.26. Write operation of a '0' on the Q node of a OR-v2 cell: detail around the rising edge of the PRE# signal.

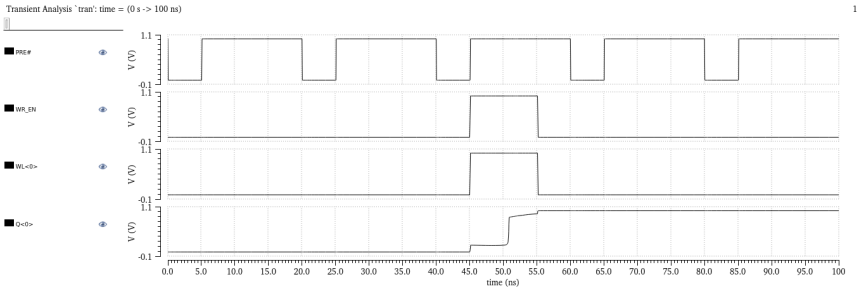

Figure 5.27. Write operation of a '1' on the Q node of a OR-v2 cell.

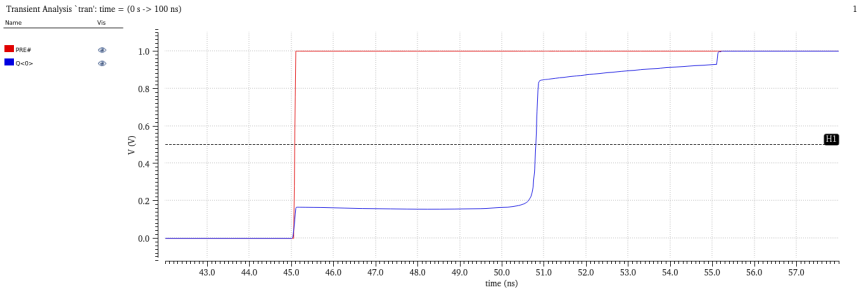

Figure 5.28. Write operation of a '1' on the Q node of a OR-v2 cell: detail around the rising edge of the PRE# signal.

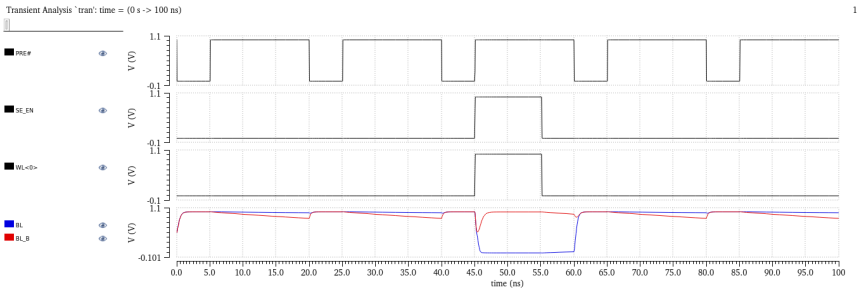

Figure 5.29. Read operation of a '0' on the BL/BL_B signals in a OR-v2 array.

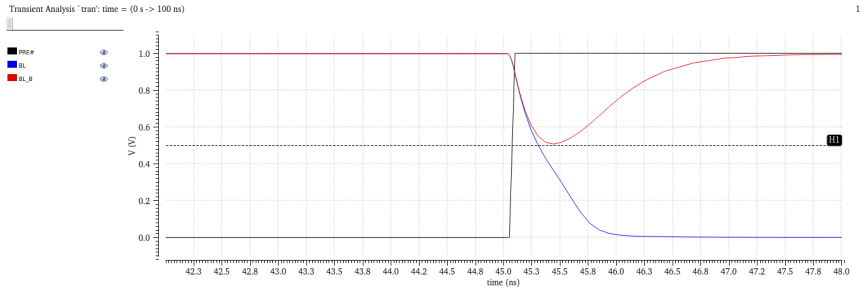

Figure 5.30. Read operation of a '0' on the BL/BL_B signals in a OR-v2 array: detail around the rising edge of the PRE# signal.

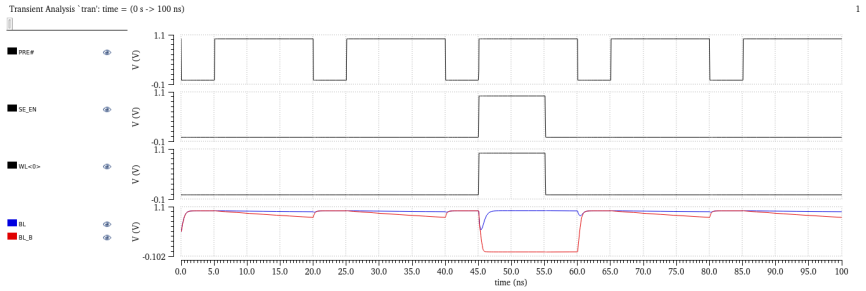

Figure 5.31. Read operation of a '1' on the BL/BL_B signals in a OR-v2 array.

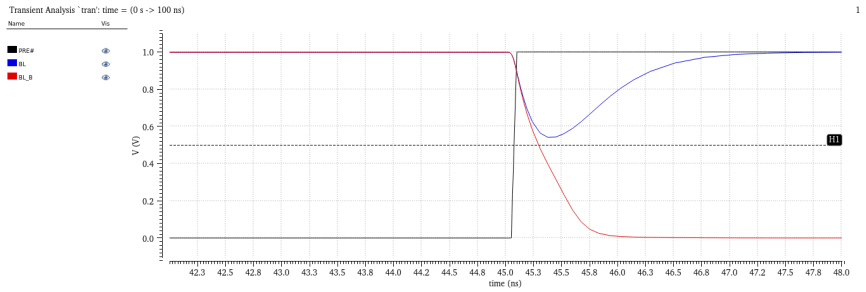

Figure 5.32. Read operation of a '1' on the BL/BL_B signals in a OR-v2 array: detail around the rising edge of the PRE# signal.

5.5 XOR

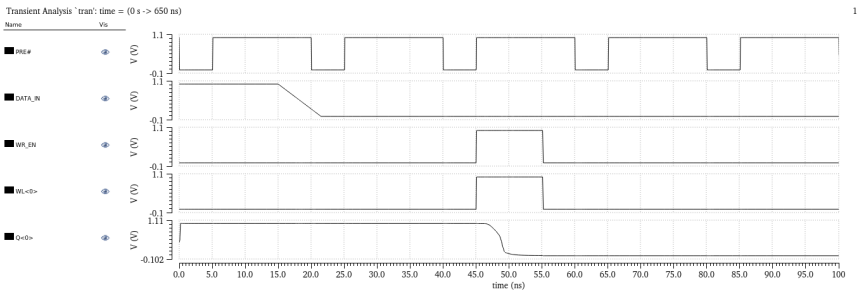

Figure 5.33. Write operation of a '0' on the Q node of a XOR cell.

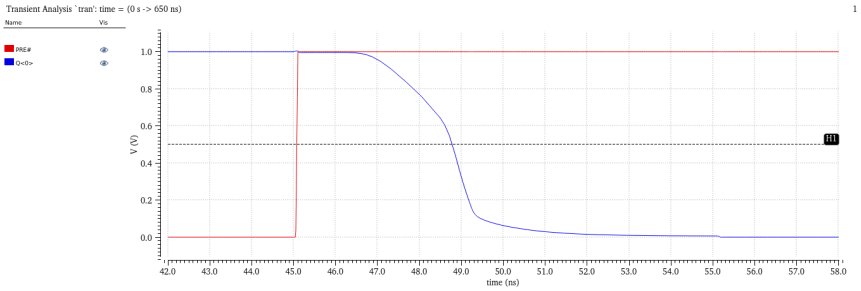

Figure 5.34. Write operation of a '0' on the Q node of a XOR cell: detail around the rising edge of the PRE# signal.

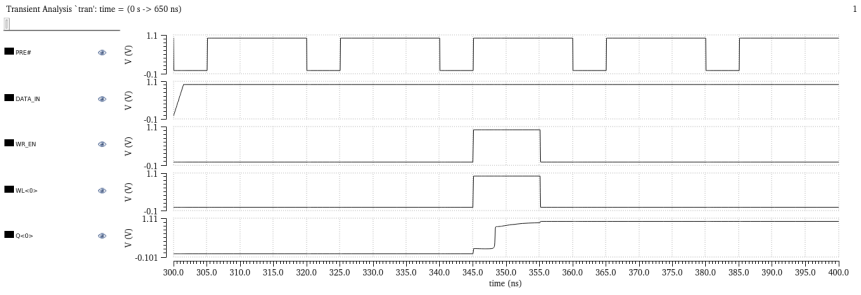

Figure 5.35. Write operation of a '1' on the Q node of a XOR cell.

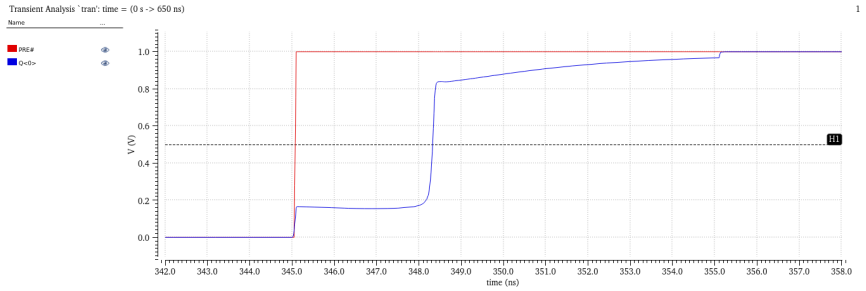

Figure 5.36. Write operation of a '1' on the Q node of a XOR cell: detail around the rising edge of the PRE# signal.

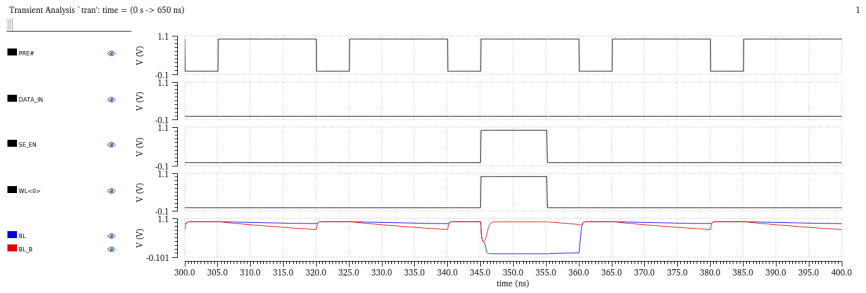

Figure 5.37. Read operation of a '0' on the BL/BL_B signals in a XOR array.

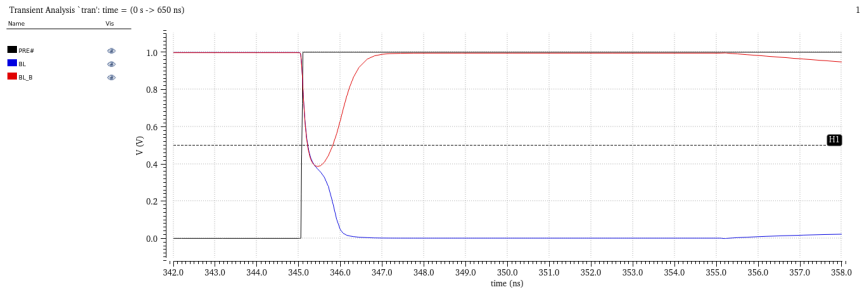

Figure 5.38. Read operation of a '0' on the BL/BL_B signals in a XOR array: detail around the rising edge of the PRE# signal.

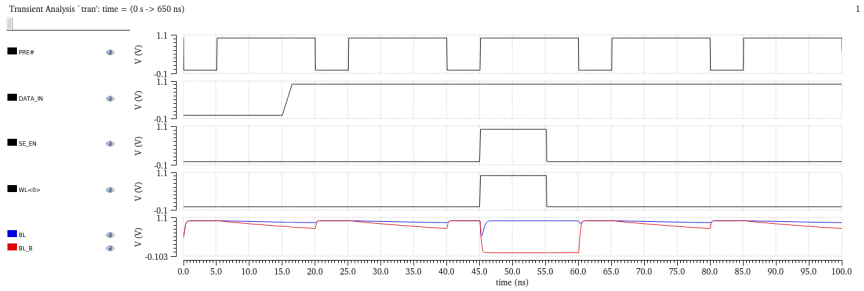

Figure 5.39. Read operation of a '1' on the BL/BL_B signals in a XOR array.

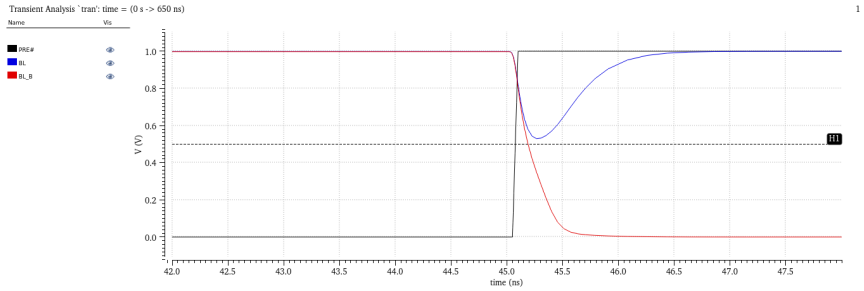

Figure 5.40. Read operation of a '1' on the BL/BL_B signals in a XOR array: detail around the rising edge of the PRE# signal.

	AND-v1	OR-v1	AND-v2	OR-v2	XOR
DELAY WR0 (ns)	6.53	6.54	6.21	6.22	3.70
DELAY WR1 (ns)	6.01	6.01	5.72	5.73	3.54
DELAY WR (ns)	6.27	6.28	5.96	5.97	3.62
DELAY SE0 (ns)	0.25	0.24	0.23	0.23	0.13
DELAY SE1 (ns)	0.23	0.23	0.22	0.22	0.12
DELAY SE (ns)	0.24	0.24	0.23	0.22	0.12

Table 5.1. Delays of the LiM-AND, LiM-OR and LiM-XOR cells.

	AND-v1	OR-v1	AND-v2	OR-v2	XOR
ENERGY WR0 (fJ)	194.50	169.00	182.90	166.80	111.00
ENERGY WR1 (fJ)	50.75	29.07	48.16	45.77	106.40
ENERGY WR (fJ)	122.62	99.03	115.53	106.28	108.70
ENERGY SE0 (fJ)	21.13	19.06	21.22	17.11	16.14
ENERGY SE1 (fJ)	18.74	17.33	16.86	15.67	13.35
ENERGY SE (fJ)	19.93	18.19	19.04	16.39	14.74

Table 5.2. Energy consumption of the LiM-AND, LiM-OR and LiM-XOR cells.

	AND-v1	OR-v1	AND-v2	OR-v2	XOR
POWER WR0 (µW)	29.78	25.84	29.45	26.82	30.00
POWER WR1 (µW)	8.44	4.84	8.42	7.99	30.06
POWER WR (µW)	19.56	15.78	19.37	17.79	30.03
POWER SE0 (µW)	85.55	77.48	91.86	74.07	119.55
POWER SE1 (µW)	80.77	74.70	76.64	71.55	117.10
POWER SE (µW)	83.24	76.13	84.43	72.84	118.43

Table 5.3. Power consumption of the LiM-AND, LiM-OR and LiM-XOR cells.

Table 5.1 show the delays of the LiM-AND, LiM-OR and LiM-XOR cells. In Table 5.2 and Table 5.3 are reported, instead, the energy and power consumption of the above mentioned LiM cells, respectively. By looking at them it is possible to make some considerations about the performances of the different layouts.
From Table 5.1 it can be observed that the "v1" layouts share almost the same values in terms of delay, and the same can be said for the "v2" layouts. The AND-v2 and OR-v2 layouts are 4.94% faster w.r.t. their "v1" counterparts for writing, and they share nearly the same sense delay. Speaking about the power, the write operation is 19.3% less consumptive adopting the AND-v2 layout and 8.15% less consumptive adopting the OR-v2 layout (w.r.t. their previous versions). It is possible to look up to a power saving also for writing, choosing the "v2" layouts: 8.54% for the LiM-AND cell and 13.72% for the LiM-OR cell.
For the sake of completeness, also the result for the layout of the LiM-XOR cell are reported, even if a comparison w.r.t. cells which implement a different logic function is not useful to the aims of this work. The cell, in fact, is used as a starting point to design the one with write-back. However, values are displayed and it is possible to look at the obtained results.

5.6 XOR-WB-v1

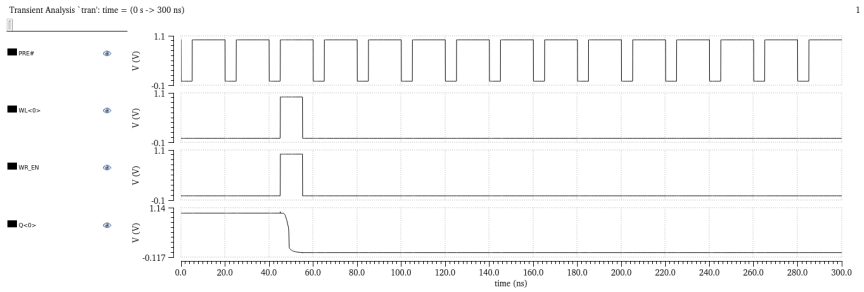

Figure 5.41. Write operation of a '0' on the Q node of a XOR-WB-v1 cell.

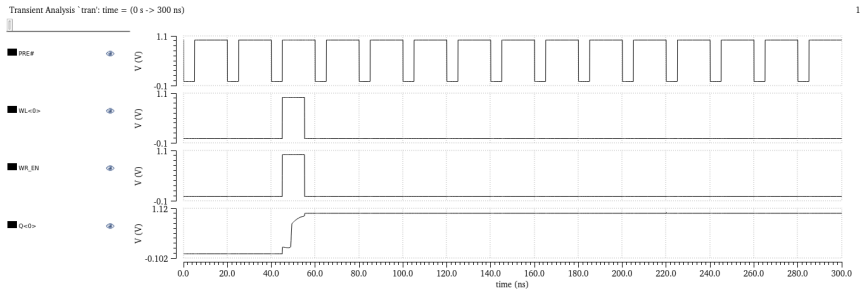

Figure 5.42. Write operation of a '1' on the Q node of a XOR-WB-v1 cell.

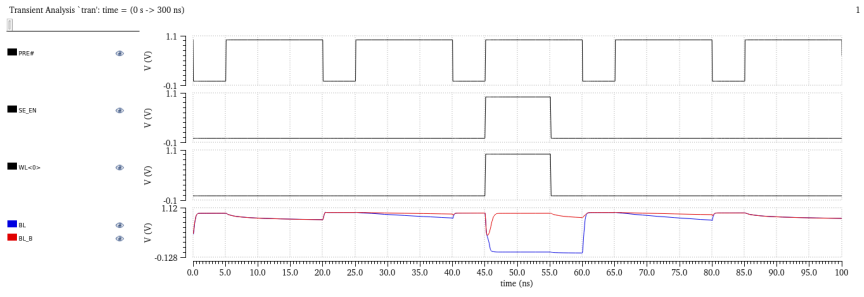

Figure 5.43. Read operation of a '0' on the BL/BL_B signals in a XOR-WB-v1 array.

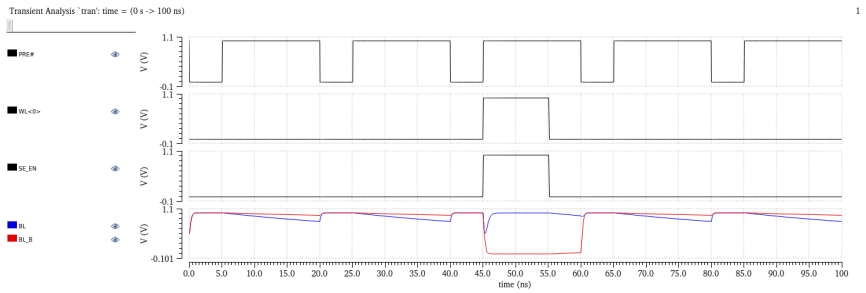

Figure 5.44. Read operation of a '1' on the BL/BL_B signals in a XOR-WB-v1 array.

Figure 5.45 and Figure 5.46 show that the cell is able to write the output of its logic gate into the storage node of the successive cell (in terms of index).

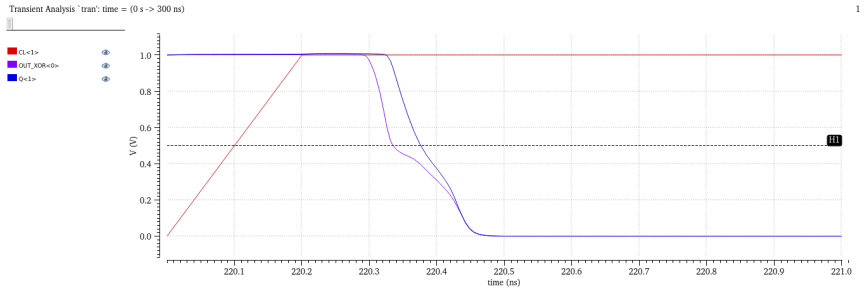

Figure 5.45. Writing the output of Cell 0 into the node Q of Cell 1: case in which a '0' is written back.

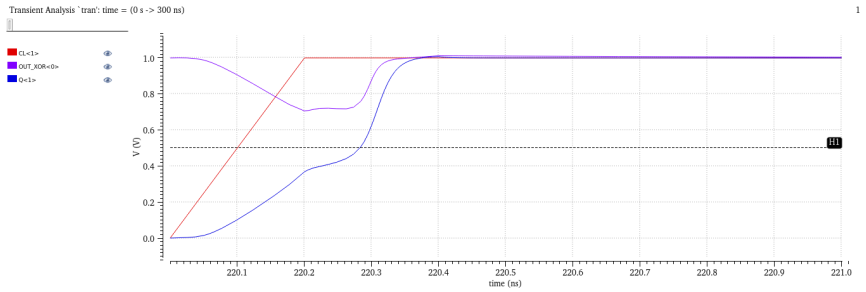

Figure 5.46. Writing the output of Cell 0 into the node Q of Cell 1: case in which a '1' is written back.

5.7 XOR-WB-v2

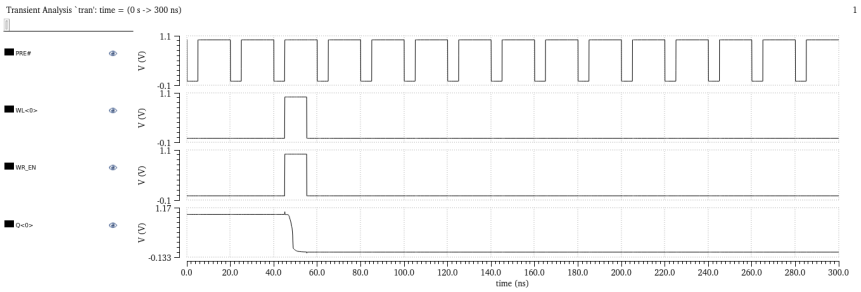

Figure 5.47. Write operation of a '0' on the Q node of a XOR-WB-v1 cell.

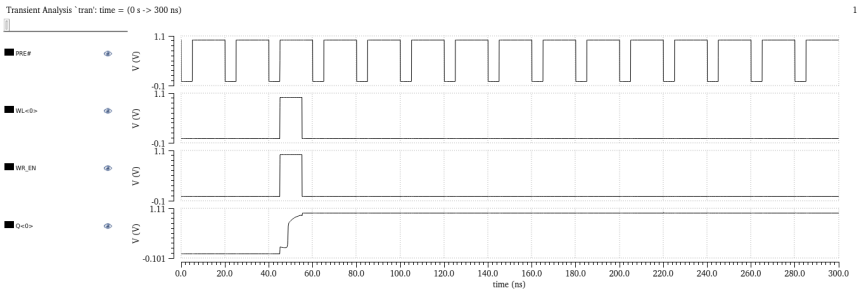

Figure 5.48. Write operation of a '1' on the Q node of a XOR-WB-v1 cell.

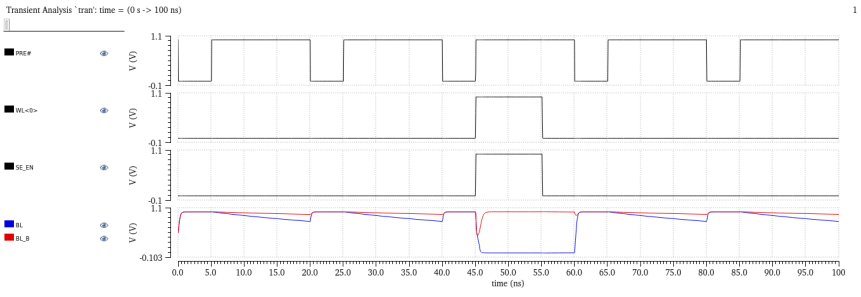

Figure 5.49. Read operation of a '0' on the BL/BL_B signals in a XOR-WB-v1 array.

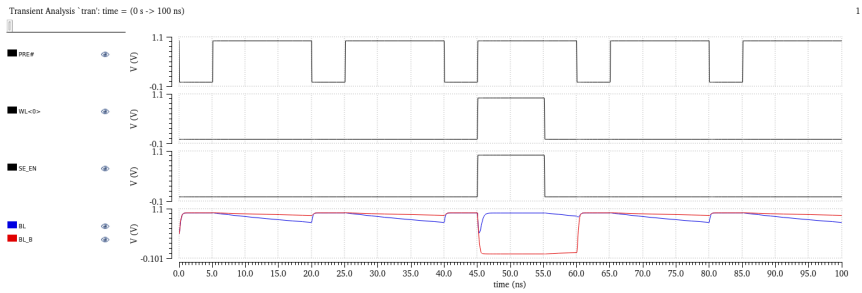

Figure 5.50. Read operation of a '1' on the BL/BL_B signals in a XOR-WB-v1 array.

Figure 5.51 and Figure 5.52 show that the cell is able to write the output of its logic gate into the storage node of the successive cell (in terms of index).

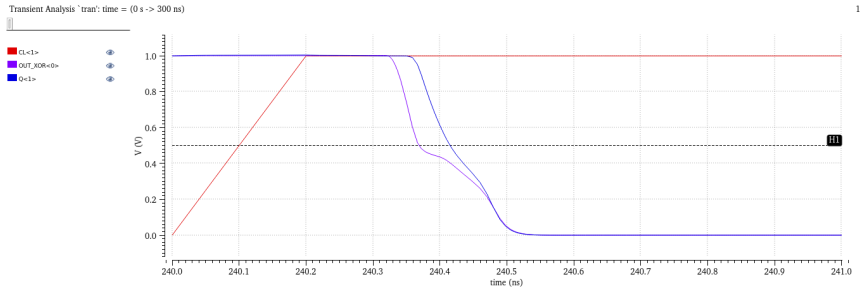

Figure 5.51. Writing the output of Cell 0 into the node Q of Cell 1: case in which a '0' is written back.

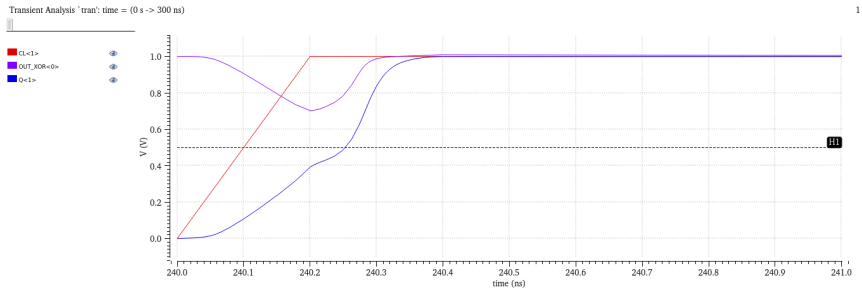

Figure 5.52. Writing the output of Cell 0 into the node Q of Cell 1: case in which a '1' is written back.

	XOR-WB1	XOR-WB2
DELAY WR0 (ns)	3.87	3.54
DELAY WR1 (ns)	4.06	3.68
DELAY WR (ns)	3.96	3.61
DELAY SE0 (ns)	0.15	0.14
DELAY SE1 (ns)	0.14	0.12
DELAY SE (ns)	0.15	0.13

Table 5.4. Delays of the LiM-XOR-WB cell.

	XOR-WB1	XOR-WB2
ENERGY WR0 (fJ)	112.80	102.80
ENERGY WR1 (fJ)	23.96	22.06
ENERGY WR (fJ)	68.38	62.43
ENERGY SE0 (fJ)	18.30	16.95
ENERGY SE1 (fJ)	16.54	14.62
ENERGY SE (fJ)	17.42	15.78

Table 5.5. Energy consumption of the LiM-XOR-WB cell.

	XOR-WB1	XOR-WB2
POWER WR0 (µW)	29.14	29.04
POWER WR1 (µW)	5.90	5.99
POWER WR (µW)	17.25	17.29
POWER SE0 (µW)	119.61	121.94
POWER SE1 (µW)	119.85	121.83
POWER SE (µW)	119.72	121.89

Table 5.6. Power consumption of the LiM-XOR-WB cell.

As previously done for the other designed cells, Table 5.4, Table 5.5 and Table 5.6 display the data about the memory operations of the LiM-XOR-WB cell. The two columns are dedicated to the two different layouts realized for the LiM cell. The second version of the layout is 8.83% faster for writing and 13.3% faster in reading; but 0.23% more power consumptive for writing and 1.81% more power consumptive for reading, w.r.t. the first one, when the LiM cell is operating as a **memory cell**.

	XOR-WB1	XOR-WB2
DELAY WB0 (ns)	0.28	0.32
DELAY WB1 (ns)	0.19	0.15
DELAY WB (ns)	0.23	0.24

Table 5.7. Delay of the write-back operation of the LiM-XOR-WB cell.

	XOR-WB1	XOR-WB2
ENERGY WB0 (fJ)	152.20	161.00
ENERGY WB1 (fJ)	28.98	19.09
ENERGY WB (fJ)	90.59	90.04

Table 5.8. Energy consumption of the write-back operation of the LiM-XOR-WB cell.

	XOR-WB1	XOR-WB2
POWER WB0 (µW)	541.16	496.91
POWER WB1 (µW)	154.97	125.59
POWER WB (µW)	387.14	378.83

Table 5.9. Power consumption of the write-back operation of the LiM-XOR-WB cell.

In Table 5.7, Table 5.8 and Table 5.9 are collected the data related to the **write-back operation**. It is possible to notice that the second version of the layout is 4.3% slower than the first one, but its adoption allow to save the 2.4% of power (and the 0.6% in terms of energy).

Chapter 6

Conclusions

After that all the data are collected, it is possible to draw some conclusions about the feasibility of the designed cells. This is done both in terms of determining the most convenient layout versions to realize, and in terms of comparing the obtained results with the one obtained for the standard 6T cell.

6.1 Area

It is possible to observe from Table 6.1 that all the LiM cells occupy more area w.r.t. the 6T cell, as expected. In fact, the addition of devices implies an area increase, due both to the device itself and to the growth of routing complexity. About the first two LiM cells, they are characterized by the same area occupation, both in their "v1" and in their "v2" implementation. In particular, AND-v1/OR-v1 has an area which is 4.2 times the one of the 6T cell. While, AND-v2/OR-v2 has an area which is only 2.5 times the one of the first cell implemented in this work. So, reasoning in terms of area, the AND-v2 and OR-v2 layouts seem to be the best choice to implement, respectively, the AND and the OR logic functions; being 1.65 times smaller w.r.t. the first layout variants.
For what concerns the LiM-XOR cell, it is obviously bigger to the previous two since it requires two more transistors and an additional input metal strip. Its area results being 5.9 times the one of the 6T cell. However, the most important cell implementing the XOR function is the LiM-XOR-WB, since it represents an improved version of the LiM-XOR. Since the additional pass-transistor needed to write-back the stored content appears

to be the largest device, both the XOR-WB-v1 and XOR-WB-v2 layouts are subject to an area increase (8.54 and 7.38 times the area of the 6T cell) Again, among the two the second variant is the one that impacts less on the area occupation.

	6T	AND-v1	OR-v1	AND-v2	OR-v2	XOR	XOR-WB-v1	XOR-WB-v2
Area (μm^2)	1.27	5.33	3.23	5.33	3.23	7.47	10.85	9.37

Table 6.1. Areas of all the designed LiM cells.

6.2 Delay

For what concerns the delays, in Table 6.2 are reported the values of the **average write delay** and the **average sense delay** (assuming an equal probability of writing/reading a '0' or a '1').
Also for what concerns delays, the AND-v2/OR-v2 layouts allow their respective LiM cells to be faster w.r.t. the adoption of the "v1" layouts. The savings in terms of time are the following: 4.78% for the write operation and 4.2 % for the read operation.
The adoption of the XOR-WB-v2 layout allows to save time: 8.4 % for writing and 7.14 % for reading, making it the best option to implement a LiM-XOR-WB cell.

	6T	AND-v1	OR-v1	AND-v2	OR-v2	XOR	XOR-WB-v1	XOR-WB-v2
T_{wr} (ns)	5.90	6.27	6.28	5.97	5.98	3.62	3.96	3.61
T_{se} (ns)	0.22	0.24	0.24	0.23	0.23	0.12	0.14	0.13

Table 6.2. Delays of all the designed LiM cells.

6.3 Consumption

The values reported in Table 6.3 (energy consumption) and Table 6.4 (power consumption) are an average computed between the '0' and '1' case of each operation. As said in section 6.2, an equal probability of handling '0's and '1's is assumed.
The AND-v2 layout is the best choice to implement the LiM-AND cell, from a consumption point of view. It allows to save 5.78 % for writing and 4.47 % for reading. Instead, the OR-v2 is more power consumptive

for writing (12.74 %) w.r.t. OR-v1, while granting a lesser dissipation for reading (4.32 %).
In terms of power dissipation, the XOR-WB-v1 layout is the most convenient choice to implement the LiM-XOR-WB cell. It allows to save 0.23 % of the dissipated power for writing and 1.81 % for reading.

	6T	AND-v1	OR-v1	AND-v2	OR-v2	XOR	XOR-WB-v1	XOR-WB-v2
E_{wr} (fJ)	66.12	122.62	99.03	115.53	106.28	108.70	68.38	62.43
E_{se} (fJ)	15.92	19.93	18.19	19.04	16.39	14.74	17.42	15.78

Table 6.3. Energy consumption of all the designed LiM cells.

	6T	AND-v1	OR-v1	AND-v2	OR-v2	XOR	XOR-WB-v1	XOR-WB-v2
P_{wr} (μW)	11.20	19.56	15.78	19.37	17.79	30.03	17.25	17.29
P_{se} (μW)	72.06	83.24	76.13	84.43	72.84	118.43	119.72	121.89

Table 6.4. Power consumption of all the designed LiM cells.

6.4 Final considerations

It is now possible to understand which IC layout is the most suitable one to implement each LiM cell discussed in this thesis work. The AND-v2 layout is the obvious choice for the **LiM-AND** cell. It is reported that it it smaller, faster and less power consumptive than the "v1". For the **LiM-OR** and **LiM-XOR-WB** cells similar considerations can be made. The OR-v2 and XOR-WB-v2 layouts are smaller and faster than the "v1" variants; they are not convenient in terms of power dissipation, though. Additionally, as seen in Table 5.9, the second variant is more convenient for the implementation of the write-back feature. So, for these last two LiM cells, the choice has to be made depending on the application, and on which metrics are considered to be the most important ones.

Finally, by looking at the collected results, it is possible to compare the values obtained for the LiM cells with the ones obtained for the 6T cell. This is a key point in order to evaluate the feasibility of the implementation of the Logic-in-Memory paradigm. In order to make the comparison between cells easier, the dissipated power of an average operation is considered (asssuming that writings and readings have an equal probability

to happen). Implementing a Lim-AND implies an area increase of 2.54 times. T_{wr} increases by 1.2 %, and P_{avg} by 24.8 %. A Lim-OR shows the same increments in terms of area and delay, but a lower one regarding the average dissipated power, which is around 8.9 %. To conclude, a LiM-XOR with the write-back feature occupies 7.38 times the area of a standard 6T cell, and the increase in terms of P_{avg} is of the 67.3 %. On the other hand, this cell grants a lowering of the T_{wr} by the 63.4 %.

In conclusion, the designed LiM cells present drawbacks in terms of area, power and delay due to the additional devices that they host, but they represent a valid solution to implement in-cell computation, due to the power and time saving that they are able to introduce at system-level, computing data directly inside the memory array. All the designed LiM cells allow parallel computation, since each cell is able to produce its own combinatorial output. The LiM-XOR-WB cell, thanks to its write-back capability, is a useful solution if the same operation have to be performed on the manipulated data multiple times (each with a different input).
As a future step, the write-back feature can be implemented to the LiM-AND and LiM-OR cells, and brand-new cells can be implemented starting from the existing ones (e.g. LiM-NAND or LiM-NOR). While, regarding the cells already presented and discussed, an interesting improvement could be made in terms of technology: a different node couldd be used instead of the employed 45 nm, in order to make LiM arrays more feasible in terms of space occupation.

Bibliography

[1] J. von Neumann, *First Draft of a Report on the EDVAC*. Moore School of Electrical Engineering University of Pennsylvania, 1945.

[2] G. Santoro, G. Turvani, and M. Graziano, "New logic-in-memory paradigms: An architectural and technological perspective," *Micromachines*, 2019.

[3] A. Coluccio, M. Vacca, and G. Turvani, "Logic-in-memory computation:is it worth it? a binary neural network case study," *Journal of Low Power Electronics and Applications*, 2020.

[4] B. Jacob, S. W. Ng, and D. T. Wang, *Memory Systems: Cache, DRAM, Disk*. Morgan Kaufmann Publishers, 2008.

[5] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits, A Design Perspective*. Prentice-Hall, 2 ed., 2003.

[6] J. Singh, S. P. Mohanty, and D. K. Pradhan, *Robust SRAM Design and Analysis*. Springer, 2013.

[7] N. H. E. Weste and D. M. Harris, *CMOS VLSI Design, A Circuit and Systems Perspective*. Addison-Wesley, 4 ed., 2011.

[8] "What is design rule checking (drc)?." [Online] https://www.synopsys.com/glossary/what-is-design-rule-checking.html, accessed 12-July-2023.

[9] "What is layout versus schematic checking?." [Online] https://www.synopsys.com/glossary/what-is-layout-versus-schematic-checking.html, accessed 12-July-2023.

[10] I. M. Copi, C. Cohen, and K. McMahon, *Introduction to Logic*. Pearson, 14 ed., 2014.

[11] W. Tang, "Fefet-based logic-in-memory supporting sa-free write-back and fully dynamic access with reduced bitline charging activity and recycled bitline charge," *IEEE Transactions on Circuits and Systems - I: Regular Papers*, 2023.