Politecnico di Torino

DIPARTIMENTO DI ELETTRONICA E TELECOMUNICAZIONI
Corso di Laurea Magistrale in Ingegneria Elettronica

Integrated Systems Technology

Register File

Group 17:

Cannavó Chiara 252944
Dongiovanni Mancino Giuseppe 245205
Motta Domenico 251312

Professors:

Piccinini Gianluca
Vacca Marco

Anno Accademico 2018/2019

Contents

List of Figures

1 Introduction

1.1 | Introduction to the laboratory experience

This project aims to estimate the characteristics of a Register File based on Traditional Planar Transistors through a MATLAB script.

The outputs of the model are:

- Delay during read and write operations
- Power consumption of a read operation
- Power consumption of a write operation
- Area and Volume

1.2 | Structure explanation

It has been chosen to analyze a structure consisting of two read ports and a write port. A typical SRAM cell is made up of six MOSFETs, two of which are pass transistors, connecting the cell to the two complementary bit lines and to the word line that activates it. Since we need to design a three port register file, we add four more pass transistor, two for each word line. Each bit in a SRAM is stored on four transistors (P1, P2, N1, N2) that form two cross-coupled inverters. This storage cell has two stable states which are used to denote 0 and 1. Each couple of pass transistor is controlled by its own word line to allow parallel reading and writing operation; we assume, however, that only one operation at a time can be performed on a single memory cell. The structure analysed includes also some *Decoders* and *Pass transistors* used to select the correct word in the correct memory block, and some *Sense Amplifier* used to recognize the value of the memory cells in a faster way. A more complete explanation is given in the Chapter 2; in fig. 1.1 is shown a block structure of this Register File, while in fig. 1.2 is shown the structure of the memory cell adapted to support two read ports and one write port.

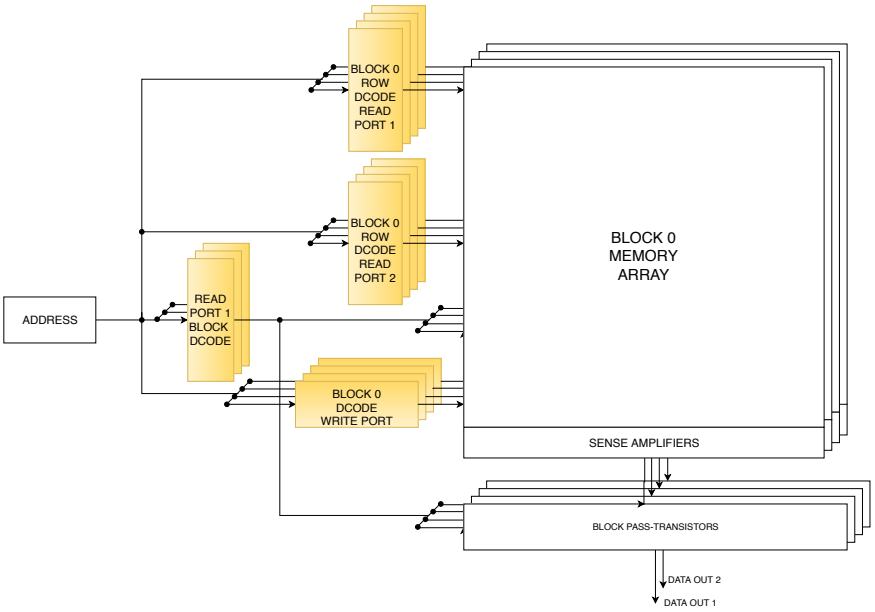

Figure 1.1: Block structure of the register file

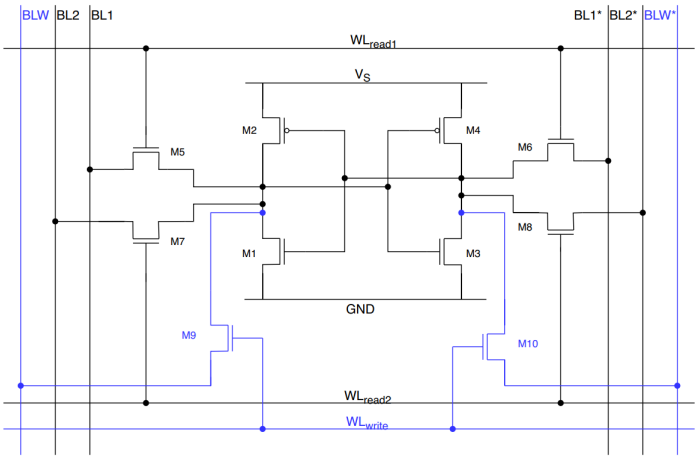

Figure 1.2: Structure of a single SRAM cell

1.3 | Behaviour of the memory

The behaviour of the memory can be described in three operation.

Precharge is the initial state to perform a read or write operation. Actually the precharged bitlines are needed only during a read operation, but the precharge operation is done at the end of each read/write cycle to simplify the control of the memory. During the precharge state, bitlines are biased at a certain voltage to speed up subsequent operations.

Read operation. During this operation, unselected blocks are electrically disconnected by the block decoder. The address of the word to be read is divided into two fields: block address and row address. In general also a column address should be needed, but since we decided to realize memory arrays with parallelism equal to the width of a single word, a column decoder is not needed. According to the active read port i (here i goes from 1 to 2, since the third and last port is a write port), the corresponding block decoder i selects the block, while all the row decoders corresponding to port i select the row; so, the block decoder and the row decoders work together. Only the row decoder corresponding to the block selected by the block decoder will be allowed to activate one of the wordlines on its output; the selected memory cells (all the ones connected to that word line) will charge the bitlines according to the stored values, the sense amplifiers between the couples of bitlines will switch, accelerating the transition, and the read value will be transmitted towards the output. A more detailed scheme of the structure of the memory, necessary for following this reasoning, is shown in fig. 1.3

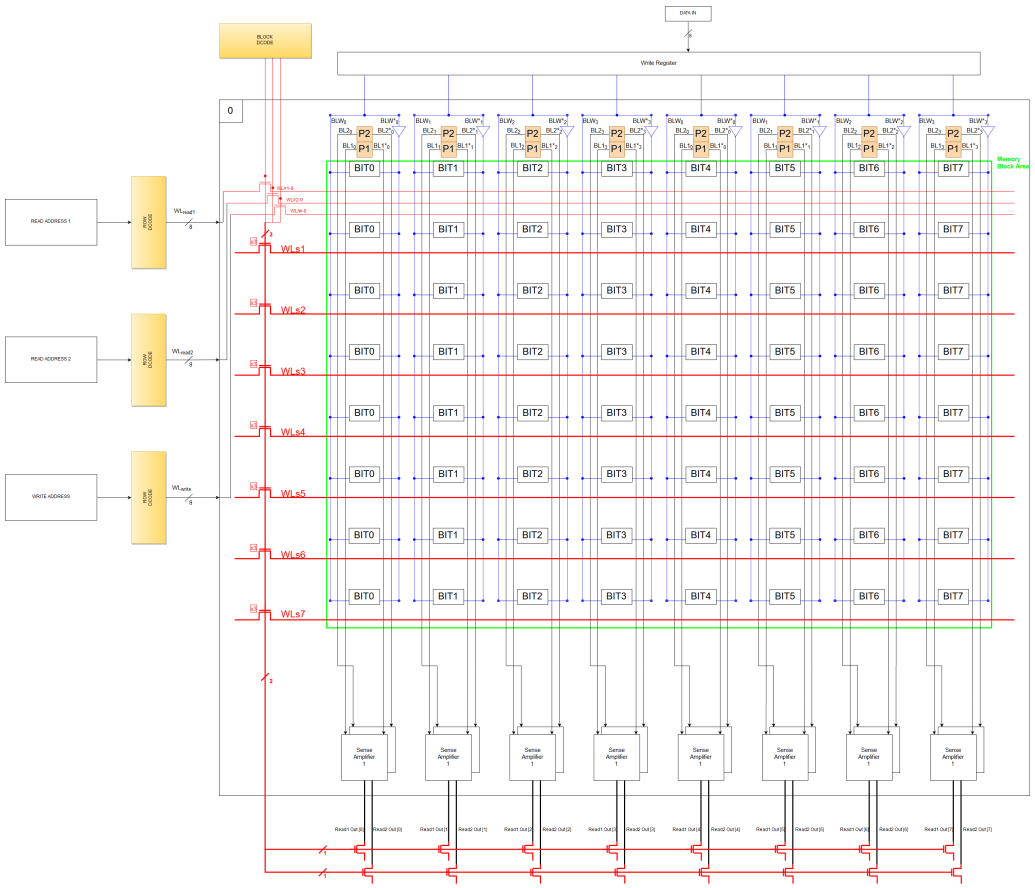

Figure 1.3: Detailed structure of the register file

Write operation. After having forced on all the bitlines in parallel the value to be written, the address of the word to be written is sent to the write block decoder and to

the write row decoders (which, again, work together with the write block decoder); only the correct row decoder will activate the write wordline of the selected word, making all the memory cells connected to it store the value present on the bitlines.

1.4 | Organization of the words

As mentioned, each memory block has a width equal to the parallelism of a single word. In particular, since generally the producers of the memory chips try to make them as square as possible, for reasons of space availability on the boards, we decided to divide the N_{word} words of parallelism N_{bit} into exactly square blocks: for this reason the number of blocks inside the register file is computed like

$$N_{block} = \left\lceil \frac{N_{word}}{N_{bit}} \right\rceil$$

where the number of rows in each block is equal to

$$N_{wl} = min(N_{word}, N_{bit})$$

Then of course the number of bits inside the block address and the row address can be computed like

$$Block\ Address = \lceil log_2(N_{block}) \rceil$$
$$Row\ Address = \lceil log_2(N_{wl}) \rceil$$

Delay 2

2.1 | Delay computation

2.1.1 | Precharge & Read operation

In the following we compute the delay due to a precharge operation followed by a read operation involving just one read port.

2.1.1.1 | Precharge unit delay

The precharge unit is driven by an *Enable* signal coming from a control unit internal to the memory (which we won't consider in this analysis). It is built by a pmos transistor for each bitline, connected from one side to the supply voltage and from the other side to the bitline, plus an equalizer pmos transistor between the two bitlines. The aim of the equalizer is to avoid that small differences in the equivalent resistance of each precharge pmos may induce some differences in the delay of the precharge operation, or equivalently in the value of the voltage reached by the two precharged bitlines.

The scheme of the unit is the following:

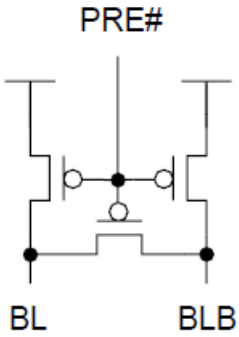

Figure 2.1: Scheme of the precharge unit

The *Enable* signal, called *PRE#* in the figure, has to discharge the gate capacitance of these three pmos, in order to make them able to switch, so the delay associated to this operation is:

$$\tau = R_{ext,pu,driver}(C_{ext,pu,driver} + 2 \cdot C_{g,pre} + C_{g,equalizer})$$

After the switching of the precharge unit, the bitline takes a certain time to be charged. This delay is computed using the Bakoglu model:

$$\tau = [R_d(C_d + C_w + C_L) + R_w C_L] + \frac{0.377}{0.69}(R_w C_w)$$

$R_d = R_{eq,pre,p}$ is the equivalent resistance of the driver, here equal to the resistance of the driving pmos transistor; $C_d = C_{s,pre}$ is the self-load capacitance of the driver, equal to the source capacitance of the same pmos transistor. $R_w = r \cdot l$ and $C_w = c \cdot l$ are the resistance and capacitance of the bitline, computed respectively from the resistance and capacitance per unit of length, where $l = BL_{length}$: here $r = BL_r$, resistance per unit of length of the bitline, and $c = BL_c + C_{d,access,l}$, where the first term is the capacitance per unit of length of the bitline, and the second term is the capacitance per unit of length due to the access transistor connecting each memory cell to the bitline. Finally, the load capacitance $C_L = (C_{d,sa,p} + C_{d,sa,n} + C_{g,sa,p} + C_{g,sa,n}) + C_{d,blockpass}$ takes into account, with the first four terms, the capacitance due to the presence of the sense amplifier connected to each couple of bitline, and with the second term the self-load capacitance of the pass transistor at the end of each bitline.

2.1.1.2 | Block decoder delay

To model the delay of this and of the other similar components we used the classical model used to determine the delay of a logic gate. The dynamic NAND decoder, in fact, is built by a precharge pmos followed by a stack of nmos transistors, as in the following picture:

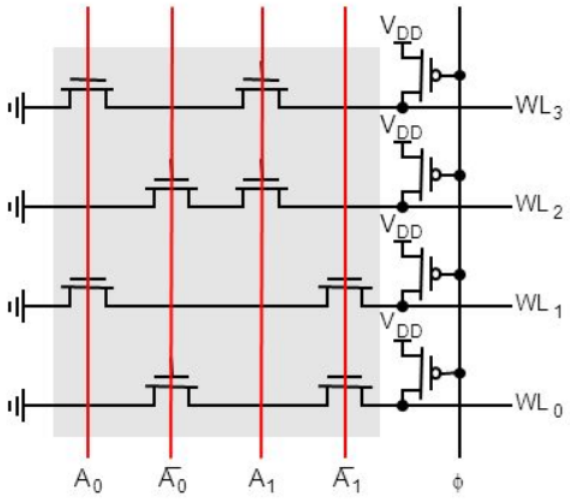

Figure 2.2: Core of the dynamic NAND decoder

Only one among these stacks of transistors will switch, making the corresponding output to go low, and so the whole structure behaves just like an ordinary logic gate.

In addition to this structure, of course, we have also the inverters placed on the input, to get also the complemented version of the address bits. Moreover, we also have a set of inverters on the output lines: the active output of a NAND decoder, in fact, is low; the output lines of the block decoder instead, as can be observed from the full scheme of the memory reported below for convenience, are needed to switch one of the nmos transistors which connect the output from the row decoder to the wordlines, and of course the nmos transistors are turned on by a high voltage.

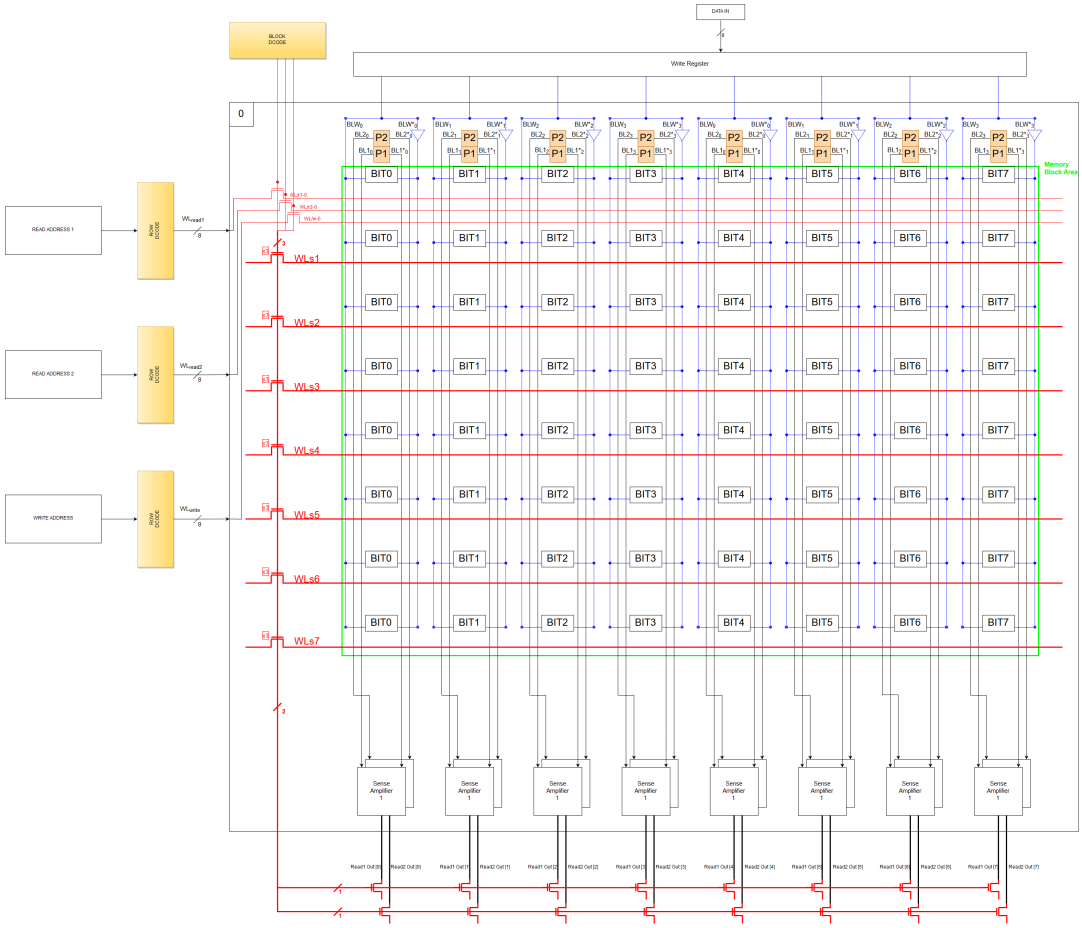

Figure 2.3: Full scheme of the memory

We consider the address bits in input to the block decoder, including their complemented version, to be stable from the beginning, while we have to take into account the delay contribution due to the inverters on the output lines.

As mentioned before, we model this delay contribution like the delay of a traditional logic gate, so:

$$\tau_{block,dec} = R_n(C_d + C_L)$$

$R_n = Stack_n \cdot R_{eq,bdec,n}$ is the equivalent output resistance due to the stack of the nmos transistors, where $R_{eq,bdec,n}$ is the output resistance of a single nmos transistor and $Stack_n = Block\_Address$ is the number of nmos transistors forming the stack. $C_d = C_{d,bdec,pcharge} + C_{d,bdec,eval}$ is the self-load capacitance due to the drain-bulk capacitance of the pmos and of the nmos on the output line. $C_L = C_{g,bdec,inv,p} + C_{g,bdec,inv,n}$ finally is the load capacitance due to the presence of the inverter on the output line.

2.1.1.2.1 | Output inverter delay The delay of the inverter on the output line is computed following the same model:

$$\tau_{block,inv} = R_p(C_d + C_L)$$

$R_p = 1 \cdot R_{eq,bdec,inv,p}$ is the output resistance of the inverter; here we have focused on the pmos because we are interested in the case in which its output is driven high, since that is the only case where it is able to switch the gate capacitance of the pass transistor it has as a load. $C_d = C_{d,bdec,inv,p} + C_{d,bdec,inv,n}$ as usual is the self-load capacitance of the gate. $C_L = N_{wl}C_{g,rowpass} + 2N_{bit}C_{g,blockpass}$ is the full load of each inverter on the output of the block decoder. This inverter in fact has to drive the N_{wl} pass transistors connected to the wordlines (row pass transistors), plus the pass transistor which allows the word just read to go out from the block (block pass transistors; hence the $+2N_{bit}C_{g,blockpass}$). Notice in fact that in the scheme we're analysing there is no column decoder; this component would be necessary if along a single row of each block of the memory there were more than one word; since we're assuming instead that each row hosts only a single word, the column decoder is useless and can be avoided.

2.1.1.3 | Row decoder delay

The next contribution is the one due to the row decoder. The block decoder and the row decoder work together, but if the number of address bits in input to the row decoder is much larger than the ones in input to the block decoder (and this is likely), also the stack of the nmos transistor will be larger and the row decoder will result to be slower than the block decoder. However, the block decoder has a load capacitance considerably higher than the one of the row decoder: not only it drives more transistors, but the capacitance to be considered in its case is the gate capacitance, which is much larger than the drain capacitance of the same transistor. So, since we don't know, at least using parametric values, which delay will be larger, we decided to compute both and to consider at the end, in the final value of the delay, only the largest between the two. The difference, as said, may be either in the contribution due to the decoder structure or in the contribution due to the driving capabilities of the inverter on the output line: for example, the block decoder may have a lower number of stack transistor, but the load of its inverter may be much larger. So, in the end, we must compare separately $\tau_{block,dec}$ with $\tau_{row,dec}$ and $\tau_{block,inv}$ with $\tau_{row,inv}$. The critical path delay will be determined by the largest from each couple of comparisons. We sum up below the structural details interested in this analysis for sake of clarity.

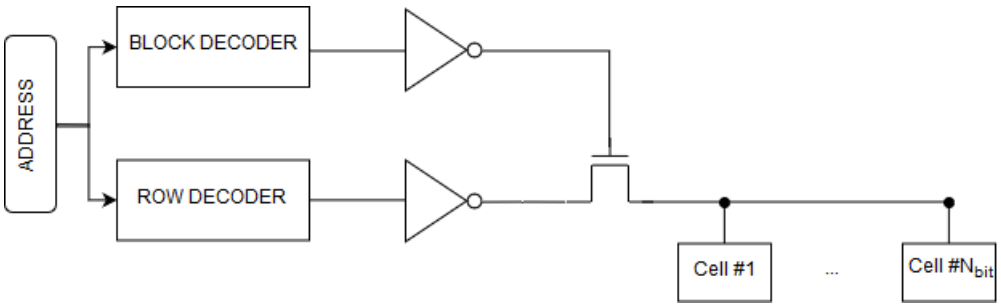

Figure 2.4: Row decoder and block decoder timing

Since the structure of the decoder is the same, also the model to compute its delay doesn't change:

$$\tau_{row,dec} = R_n(C_d + C_L)$$

$R_n = Stack_n \cdot R_{eq,rdec,n}$ is the equivalent resistance due to the stack of the nmos transistors, and $Stack_n = Row\_Address$. $C_d = C_{d,rdec,pcharge} + C_{d,rdec,eval}$ is the self-load capacitance. $C_L = C_{g,rdec,inv,p} + C_{g,rdec,inv,n}$ is again the load capacitance due to the inverter on the output.

2.1.1.3.1 | **Word line delay** The inverter on the output of the row decoder is taken into account in this section, since it works as driver for the charge of the selected word line. Due to the presence of the pass transistor between the row decoder and the word line, which represents the load to be charged, we have to use the Elmore model to represent the situation.

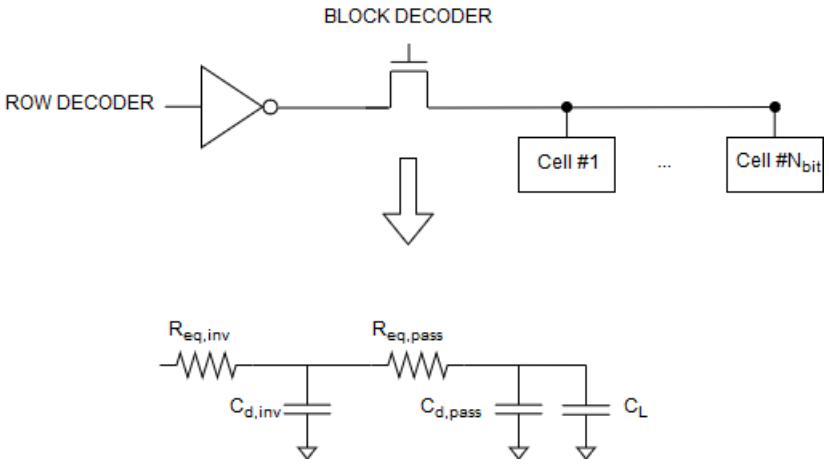

Figure 2.5: Elmore model for the wordline delay

Each memory cell is connected to the wordline through the gate of the two access transistors. So, the load C_L to be charged can be computed as $C_L = 2C_{g,access,l} \cdot WL_{length}$, where $C_{g,access,l}$ is the gate capacitance of a single access transistor per unit of length and WL_{length} is the length of the wordline.

The equation to compute the delay with the Elmore model becomes:

$$\tau_{row,inv} = R_{eq,inv}(C_{d,inv} + C_{d,pass} + C_L) + R_{eq,pass}(C_{d,pass} + C_L)$$

$R_{eq,inv} = R_{eq,rdec,inv,p}$ is the equivalent resistance from the output of the inverter (again, we focus on the pmos because the interesting case is when its output goes high). $C_{d,inv} = C_{d,rdec,inv,p} + C_{d,rdec,inv,n}$ is the self-load capacitance of the inverter. $R_{eq,pass} = R_{eq,rowpass}$ and $C_{d,pass} = C_{d,rowpass}$ are respectively the equivalent resistance and drain capacitance of the pass transistor which drives the wordline. $C_L = 2C_{g,access,l} \cdot WL_{length}$ is the total load capacitance for each wordline, as described above.

Before analysing the delay due to the reading of the value stored inside the cell, we insert a brief review about the functioning of the SRAM memory cell, and the constrains we have on the transistors size.

2.1.1.4 | SRAM memory cell

The structure of the cell, assuming to have two read ports and one write port, is the following:

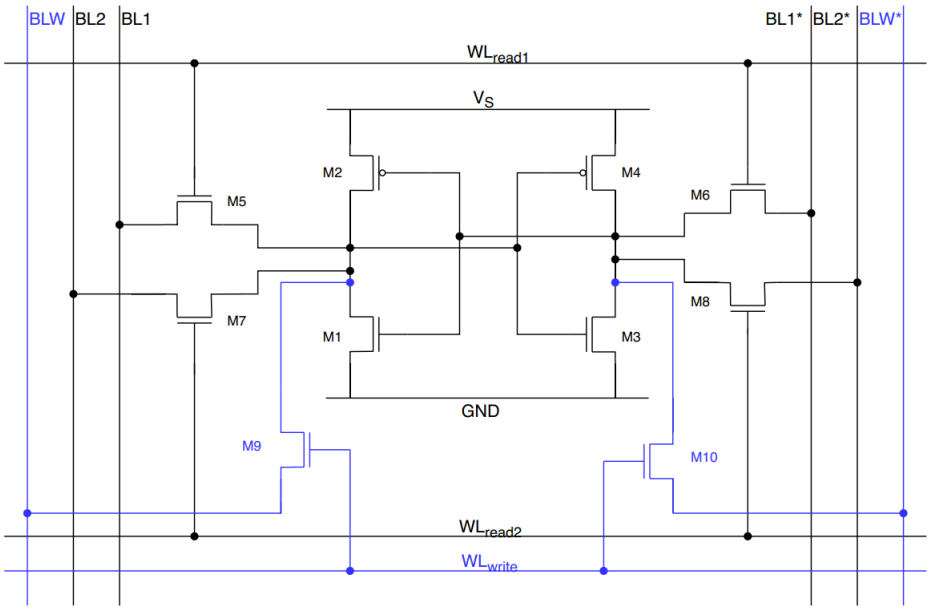

Figure 2.6: Structure of a SRAM cell

Let's say we want to read with the *port1* a value 0 stored in the cell: then M1 and M4 are turned on, while M2 and M4 are off. After precharging both bitlines to *Vdd* and activating WL_{read1}, *BL1* will be discharged through the series of M1 and M5 (remember that, to avoid errors, we must have that $W_1 > W_5 > W_2$). So, the huge bitline capacitance (capacitance per unit of length including BL_c and the capacitance per unit of length due to the access transistors, plus the input capacitance of the sense amplifier and the drain capacitance of the column pass transistor) will be discharged through the series of those two transistors. However, thanks to the sense amplifier, we don't need to wait the full discharge, because after only few percents of the full *Vdd* voltage swing the sense amplifier will go out its metastable state and will accelerate the transition. We indicate these "few percents" with the parameter K_{SA}.

If instead we are trying to write, let's say, a 1 into the memory cell, while for example the stored value is 0 (so, M1 and M4 on, M2 and M3 off): first of all we must force the value 1 and 0 respectively on BLW and BLW^* with an external driver. M9 and M10 will be turned on by WL_{write}: since M4 is more resistive than M10, the node between M4 and M10 will be discharged through the resistance of M10; as soon as the voltage on that node goes below the threshold of the inverter composed by M1 and M2, M2 will turn on while M1 turns off, so that also the node connected to M9 switches from 0 to 1 and also the value previously imposed on BLW is correctly stored inside the cell.

2.1.1.4.1 | Bit line delay So, during the read operation, the memory cell acts as a driver on a huge capacitance. However, as said, we can consider only a portion K_{SA} of the whole delay thanks to the sense amplifier.

We can compute the delay due to the read operation applying the Bakoglu model:

$$\tau = [R_d(C_d + C_w + C_L) + R_w C_L] + \frac{0.377}{0.69}(R_w C_w)$$

$R_d = R_{eq,cell,n} + R_{eq,access,n}$ is the equivalent resistance of the driver, which this time is the memory cell; $C_d = C_{d,access}$ is the capacitance associated to the driver, equal to the drain capacitance of the access transistor. $R_w = r \cdot l = BL_r \cdot BL_{length}$ is the total resistance of the bitline, while $C_w = c \cdot l = (BL_c + C_{d,access,l})BL_{length}$, like in the case of the precharge. Finally, the load capacitance is again $C_L = (C_{d,sa,p} + C_{d,sa,n} + C_{g,sa,p} + C_{g,sa,n}) + C_{d,blockpass}$.

Including also the K_{SA} factor, the equation becomes:

$$\tau = K_{SA}\{[R_d(C_d + C_w + C_L) + R_w C_L] + \frac{0.377}{0.69}(R_w C_w)\}$$

2.1.1.5 | Sense amplifier delay

The sense amplifier is made with two cross coupled inverters that are brought to a metastable state and then are applied a voltage difference by means of the input bitlines. Notice that in this amplifier input and output are somehow coincident. The schematic of the component is shown below:

Its delay is described by a simple RC model:

$$\tau = R_{eq,sa,mod,parallel}[C_{d,sa,p} + C_{d,sa,n} + C_{g,sa,p} + C_{g,sa,n} + (BL_c \cdot BL_{length}) + C_{d,blockpass}]$$

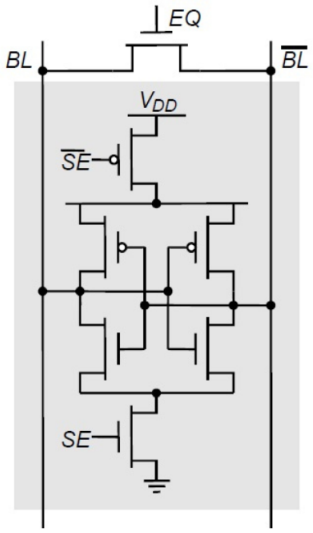

Figure 2.7: Sense amplifier structure

In this equation we take into account the equivalent resistance of the sense amplifier, which drives its self-load capacitance, the capacitance due to the gate of the cross-coupled inverter, the capacitance of the bitline, plus the drain capacitance of the pass transistor connected at the bottom of the bitlines and whose gate terminal is driven by the block decoder.

2.1.1.6 | Delay of the column pass transistor

Finally, the last contribution to the delay is given by the pass transistor (one for each bitline) we have before the output from the block. It is driven by the block decoder (by the inverter on its output). To estimate the delay associated to this component is enough to compute:

$$\tau = R_{eq,pass}(C_{d,pass} + C_L)$$

$R_{eq,pass} = R_{eq,colpass}$ is the equivalent resistance of the pass transistor, whereas $C_{d,pass} = C_{d,blockpass}$ is its self-load capacitance. C_L is unknown and in our analysis is assumed to be an open circuit.

2.1.1.7 | Total delay

The total delay is computed as the sum of all the contributions described up to now (with the exception of the block decoder and the row decoder, as already described), multiplied by 0.69.

2.1.2 | Write operation

Having already analysed all the contributions to the delay associated to a read operation, analysing the delay of a write operation is very simple. First of all, we give as input

to the block and to the row decoder, both specific for the write operation, the address of the row where we want to write the word provided on input of the memory. The row decoder will activate only one of the pass transistors on its output lines, so that one of the N_{wl} wordlines dedicated to the write operation will be activated. At the same time, an external driver must load the N_{bit} bit to be written on the bitlines. Finally, as soon as the concerned WL_{write} switches, the access transistors connected to it will turn on, allowing the writing of the bits into the cells along the selected row.

So, the only differences are: the delay associated to the need to put on the bitlines the value to be stored (which however is a contribution very similar to the precharge operation; in particular, we don't consider the precharge contribution to the writing delay. The precharge before the write operation is done just to simply the control of the memory, unlike the case of a read operation, where the precharge is needed for real. So here the driver delay conceptually substitutes the precharge delay), and the delay associated to the writing of the bit inside the memory cell, operation already detailed in section 2.1.1.4. A little additional difference comes from the contribution $\tau_{block,inv}$: this time in fact the load capacitance is only $C_L = N_{wl}C_{g,rowpass}$, we don't have the $+2N_{bit}C_{g,blockpass}$ term anymore.

2.1.2.1 | Driver delay

We assume that the value of the data to be stored is forced on the bitlines before the address bits are provided in input to the decoders. In this way, the contribution due to this operation conceptually replaces the precharge operation. In principle the two operations could even happen at the same time: the driver forces the correct values on the bitlines while the decoders are still interpreting the address bits and driving their output lines consequently; however, in this second case, for data intergity reasons, we would need that $\tau_{row,dec} + \tau_{row,inv} > \tau_{driver}$ and so a very firm control on the process, because otherwise we may write a wrong value into the row: so, it is quite a dangerous assumption to do.

We can again model this contribution with Bakoglu, just like we have done in 2.1.1.1.

$$\tau = [R_d(C_d + C_w)] + \frac{0.377}{0.69}(R_w C_w)$$

$R_d = R_{eq,driver}$ and $C_d = C_{driver}$ are the equivalent resistance and capacitance of the external driver. $R_w = r \cdot l = BL_r \cdot l$ and $C_w = c \cdot l = (BL_c + C_{d,access,l}) \cdot l$ are the resistance and capacitance of the bitline. Here we don't have any "lumped" load capacitance C_L.

2.1.2.2 | Cell delay

As already described in section 2.1.1.4, we assume to be writing a 1 into a memory cell currently containing a 0. Due to the constrains on the width of the transistors internal to the cell ($W_1 > W_9 > W_2$ and $W_3 > W_{10} > W_4$), we only able to write a 0, not a 1, so we have to consider first the right side of the cell, related to BLW^*. The transistor M_4 is on, the bitline BLW^* is hosting the value 0, and M_{10} is trying to discharge the node it has in common with M_4. The situation then can be summed up like:

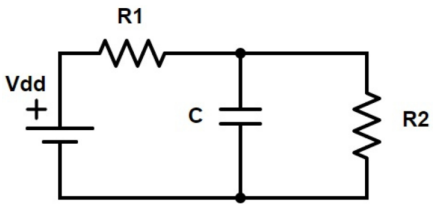

Figure 2.8: Equivalent circuit for the first half of the write delay

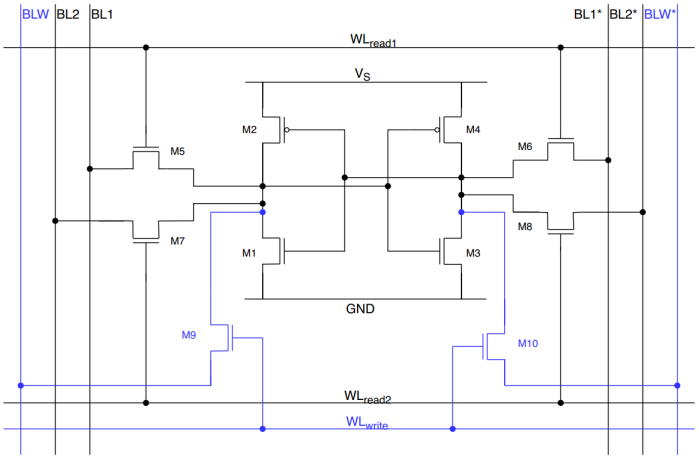

Figure 2.9: SRAM cell

R_1 and R_2 are related respectively to M_4 and M_{10} (we report the same picture of the memory cell already shown for convenience), and $R_1 > R_2$. C is the capacitance of the node between the two transistors and is equal to $C_{g,cell,p} + C_{g,cell,n} + C_{d,cell,p} + C_{d,cell,n} + C_{s,access}$. The time constant of this circuit is equal to

$$\tau_1 = \frac{R_1 R_2}{R_1 + R_2} \cdot C$$

This means that the capacitor C will be charged with this time constant. As soon as the voltage on the node goes above the threshold of the inverter composed by M_1 and M_2, the inverter will take a certain time to switch, while charging the capacitance related to the node in common between the access transistor M_9, M_1 and M_2. Only at this point the value is correctly stored inside the cell and the operation ends. This "certain time" can be computed like:

$$\tau_2 = R_{eq,cell,p} \cdot C$$

$R_{eq,cell,p}$ is the resistance of the pmos transistor M_2, while C is symmetrical to the capacitance represented in the picture above and has the same expression.

Since we don't know the value for the threshold of the inverter, we assume that this value is reached after $4\tau_1$. Then, the total delay associated to the writing of the value inside the memory cell is equal to

$$\tau = 4\tau_1 + \tau_2$$

2.2 | Simulation results

To verify the plausibility of our computations we assigned a reasonable value to each of the parameters involved in the equations. We also made the number of bits, contained in each word hosted in the memory change, in a well defined range, in order to analyse

how the delay changes if we vary the dimensions of the memory array. In particular, we considered $N_{word} = 128$ and N_{bit} in the range $[8, 16, 32, 64, 128]$.

The results we obtained are reported in the figures below. In particular, in fig. 2.10 is shown the delay we have with a read operation. We can notice a more than linear behaviour with the increase of N_{bit}. The reason for this is in the usage of the Bakoglu model in sections 2.1.1.1 and 2.1.1.4.1. In particular, the distributed term inside the Bakoglu equation involves a BL^2_{length} contribution, and we must remember that BL_{length} depends linearly on the variable N_{bit}. To verify our reasoning we commented the lines of code involving the Bakoglu delay computation and we found a linear behaviour, as we expected.

We can also notice that, with the choice of values we have made, the expected value for the read delay spans between $0.16ps$ for $N_{bit} = 8$ and $12.4ps$ for $N_{bit} = 128$.

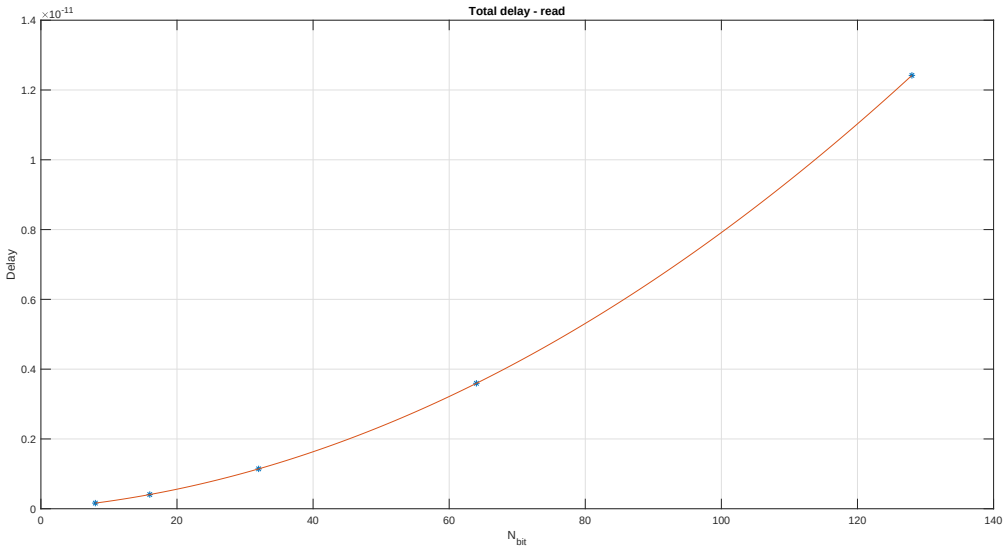

Figure 2.10: Delay on a read operation

In fig. 2.11 we report the simulation concerning the delay on the write operation. As already said, in the delay computation of a write operation, the precharge is substituted by the writing of the values to be stored on the bitlines; the block and the row decoder contributions remain the same, with the only difference that here the load of the inverter on the output of the block decoder is a bit lower; finally, we have a different delay associated to the writing of the value inside the memory cell, while we don't have the contributions of the sense amplifier and of the pass transistor at the end of the bitlines. So, overall, we expect to have a result a bit lower with respect to the delay associated to the read operation.

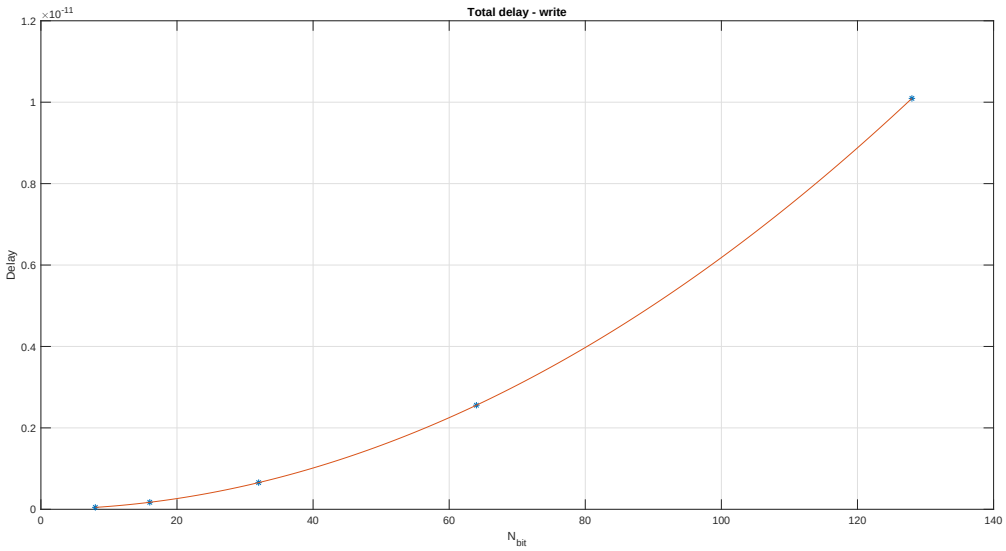

Figure 2.11: Delay on a write operation

We can observe again a more than linear behaviour, due to the use of the Bakoglu model in sec. 2.1.2.1. For $N_{bit} = 8$ we have a delay of $0.047ps$ (after all, in this case we have a register file divided in 16 blocks of 8 words with 8 bits each, so each memory array is very small), while for $N_{bit} = 128$ we have a delay of about $10ps$.

3 Power Analysis

The following analysis is based on the assumption that only one operation at time is carried out. The power consumption in the case of contemporary operations, being the register file a multi-port device, will be equal to the sum of the single contributions.

3.1 | Capacitance modeling

The considered capacitances are the following:

- $C_{g,access}$ gate capacitance of a generic pass transistor
- $C_{d,access}$ "drain/source" capacitance of a generic pass transistor
- $C_{g,rowpass}$, $C_{g,blockpass}$ gate capacitance of internal and external block pass transistors
- $C_{d,rowpass}$, $C_{d,blockpass}$ drain capacitance of internal and external block pass transistors
- $C_{wl,wire}$ wire capacitance per unit length of wordline
- $C_{bl,wire}$ wire capacitance per unit length of bitline
- $C_{block,dec}, C_{row,dec}$ total capacitances of each type of decoder
- $C_{SA,in}$ input capacitance of the sense amplifier
- $C_{g,pre}$, $C_{s,pre}$ gate and source capacitance of the precharge transtistor
- C_{ext,pu_driver} external precharge unit driver capacitance
- $C_{g,equalizer}$ gate capacitance of the equalizer mos

Other useful parameters are:

- N_{bit} number of bit for each row, equal to the number of column;
- N_{wl} number of wordlines

- BL_{length} length of bitline
- WL_{length} length of wordline
- N_{block} number of block
- N_{erase} number of erase cycles

3.1.1 | Precharge block capacitances

To evaluate power dissipation of the precharge block, the following capacitance is considered:

$$C_{prec} = C_{ext,pu\_driver} + 2 \cdot C_{g,pre} + C_{g,equalizer}$$

3.1.2 | Lines capacitances

To accurately evaluate the capacitance of the lines (C_{bl} and C_{wl}), the capacitance of the wires and the length of bitline/wordline are considered. So the total capacitance for each line is:

$$C_{wl} = C_{d,rowpass} + 2 \cdot C_{g,access} \cdot N_{bit} + WL_c \cdot WL_{length}$$

$$C_{bl} = C_{s,pre} + C_{d,access} \cdot N_{wl} + BL_c \cdot BL_{length} + C_{SA,in}$$

$$C_{bl,write} = C_{d,driver} + C_{d,access} \cdot N_{wl} + BL_c \cdot BL_{length}$$

The capacitance $C_{bl,write}$ is similar to the C_{bl}, but without the contribution of the precharge block and the sense amplifier capacitances and with a driver load.

where $C_{SA,in}$ is the input capacitance of the sense amplifier

3.1.3 | Decoders capacitance

To evaluate the capacitance of the decoders, the following formulas are used:

$$C_{block,dec} = C_{d,bdec,pcharge} + C_{d,bdec,eval} + C_{g,bdec,inv_p} + C_{g,bdec,inv_n}$$

$$C_{block,stack} = 0.5 \cdot C_{g,bdec,n} \cdot Block\_Address \cdot N_{block};$$

where $C_{g,dec,eval}, C_{d,bdec,pcharge}$ and $C_{d,bdec,eval}$ are, respectively, the gate/drain capacitance of the precharge and evaluation transistors, while the others are the gate capacitances of the output inverter. In particular, $C_{d,bdec,eval} = Block\_Address \cdot C_{d,bdec,n}$.
$C_{block,stack}$ is the equivalent gate capacitance, that has to be loaded to turn on the n-mos and select the correct output, given certain selection bits from the address.
Similar expressions have been used for row decoder.

3.1.4 | Sense Amplifier

To evaluate the capacitance of the sense amplifier, the following formulas are used:

$$C_{SA,in} = C_{d,sa,p} + C_{d,sa,n} + C_{g,sa,p} + C_{g,sa,n}$$

3.2 | Read power

3.2.1 | Decoding stage

During a read operation, the block decoder selects the block in which the operation has to be carried out. At the same time the row decoder is working to select the addressed word. The formulas to calculate energy consumption in the decoding phase are the following:

$$E_{block,dec} = 0.5 \cdot C_{block,dec} \cdot V_{on,pt}^2$$

$$E_{stack,bdec} = 0.5 \cdot C_{block,stack} \cdot V_{on,pt}^2$$

$$E_{row,dec} = 0.5 \cdot C_{row,dec} \cdot (V_{sel}^2 + V_{unsel}^2 \cdot (N_{wl} - 1)) \cdot N_{block}$$

$$E_{stack,rdec} = 0.5 \cdot C_{row,stack} \cdot V_{on,pt}^2 \cdot N_{block}$$

The energy consumption linked to the selection of the correct block:

$$E_{row,pt} = 0.5 \cdot (C_{g,rowpass} \cdot N_{wl} + C_{g,blockpass} \cdot N_{bit}) \cdot V_{on,pt}^2$$

3.2.2 | Precharge

To speed up the following operations, in order to reduce the latency, the two bitlines are biased to a specific value $V_{bl,prec}$, by the precharge block. Being $V_{bl,prec}$ the final voltage after the equalization between the BL and the complementary one, the starting voltage of the precharge is two times $V_{bl,prec}$. The other lines are supposed to be biased to ground. The formulas to calculate dissipated energy in the precharge phase are the following:

$$E_{pre} = 0.5 \cdot C_{prec} \cdot (V_{on,pt}^2) \cdot N_b l$$

$$E_{bl} = 0.5 \cdot C_{bl} \cdot V_{bl,prec}^2 \cdot N_{bit}$$

3.2.3 | Read operation

The wordline of the selected word is biased to $V_{on,pt}$, in order to make the memory cell accessible, while the voltage on the unselected words is set to ground. It is assumed that the initial voltage of the wordline is 0.

The energy to switch the selected and the unselected wordlines is the following:

$$E_{sel} = 0.5 \cdot C_{wl} \cdot (V_{sel}^2 + V_{unsel}^2 \cdot (N_{wl} - 1))$$

where $V_{on,pt}$ is the voltage to enable the memory cell pass transistors.
The bitlines are at the precharge voltage $V_{bl,prec}$ and can have two different kinds of voltage drop, $V_{rd,1}$ and $V_{rd,0}$. Being the two bitlines identical, the energy to read is:

$$E_{read} = 0.5 \cdot C_{bl} \cdot ((V_{bl,prec} - V_{rd,0})^2 + (V_{bl,prec} - V_{rd,1})^2) \cdot N_{bit}$$

where $V_{bl,prec} - V_{rd,0}$ and $V_{bl,prec} - V_{rd,1}$ are the voltage drops to read '0' and '1'.

3.2.4 | Sense Amplifier

The state change in the bitline is detected using a sense amplifier connected to each line. The energy consumption related to this stage is given by:

$$E_{SA} = 0.5 \cdot C_{d,blockpass} \cdot V_{bl,prec} \cdot ((V_{bl,prec} - V_{rd,0}) + (V_{bl,prec} - V_{rd,1})) \cdot N_{bit}$$

3.2.5 | Total Dynamic Read Power

Total read energy is computed as summation of all the previous terms. Assuming f_{read} as read frequency and E_{read}as total read energy, total read power can be computed as follows:

$$P_{read} = E_{read} \cdot f_{read}$$

3.3 | Write Dynamic Power

Similar analysis can be done for the write operation. Having the same decoding stage to select the addressed location, same relations can be used: a little difference is present in the $E_{row,pt}$. During the write operation, there is no output transistor selected by the write decoder, as Figure 2.3 shows.

$$E_{row,pt} = 0.5 \cdot (C_{g,rowpass} \cdot N_{wl}) \cdot V_{on,pt}^2$$

To write a certain word is necessary to load through an external driver the N_{bit} bit and charge the respective bitlines to that value. As previously discussed, this step follows the precharge: the driver forces the correct values on the precharged bitlines, discharging one of the two.
Basically, one bitline is charged to a certain voltage and the other one is charged to its complementary voltage. Defining V_{prog} and V_{unprog} respectively as voltage to write a '1' and its complement, the used relation is the following:

$$E_{prog} = 0.5 \cdot C_{bl,write} \cdot (V_{prog}^2 + V_{unprog}^2) \cdot N_{bit}$$

Total write energy is so computed as summation of all the previous terms. Assuming f_{read} as read frequency and E_{read}as total write energy, total write power can be computed as follows:

$$P_{write} = E_{write} \cdot f_{write}$$

3.4 | Simulation results

As for the delay, to verify the plausibility of our computations we assigned a reasonable value to each of the parameters involved in the equations.
The obtained results are reported in the figures below.

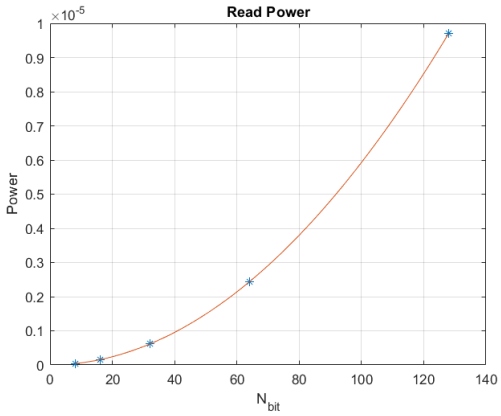

Figure 3.1: Dynamic Power Consumption on a read operation

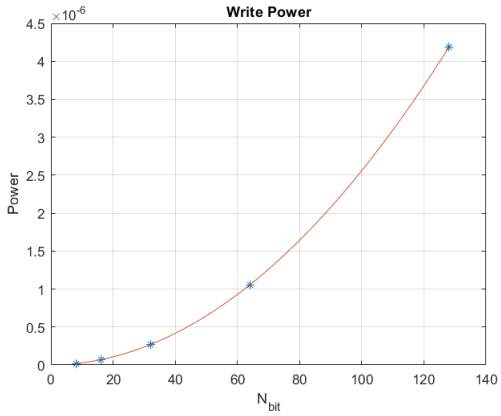

Figure 3.2: Dynamic Power Consumption on a write operation

The behaviour represented is reasonable: in all the contributions previously discussed there is a more than linear dependence on N_{bit}. In particular the most relevant contributions are dependent from the product of N_{bit} and BL_{length} or WL_{length}, that depend linearly by the first one, giving as result a quadratic curve.
The curves above show that power consumption for read and write operations is in the order of some μW, with a slightly larger value for the read operation, due to the power consumption of the sensing step.

4 Area and Volume

In this chapter we have computed the area and the volume of the complete structure of the register file (10T Cell, Decoders and Sense Amplifier).

4.1 | Memory Block

For the area and volume evaluation of the memory, the model in figure 4.1 has been used.

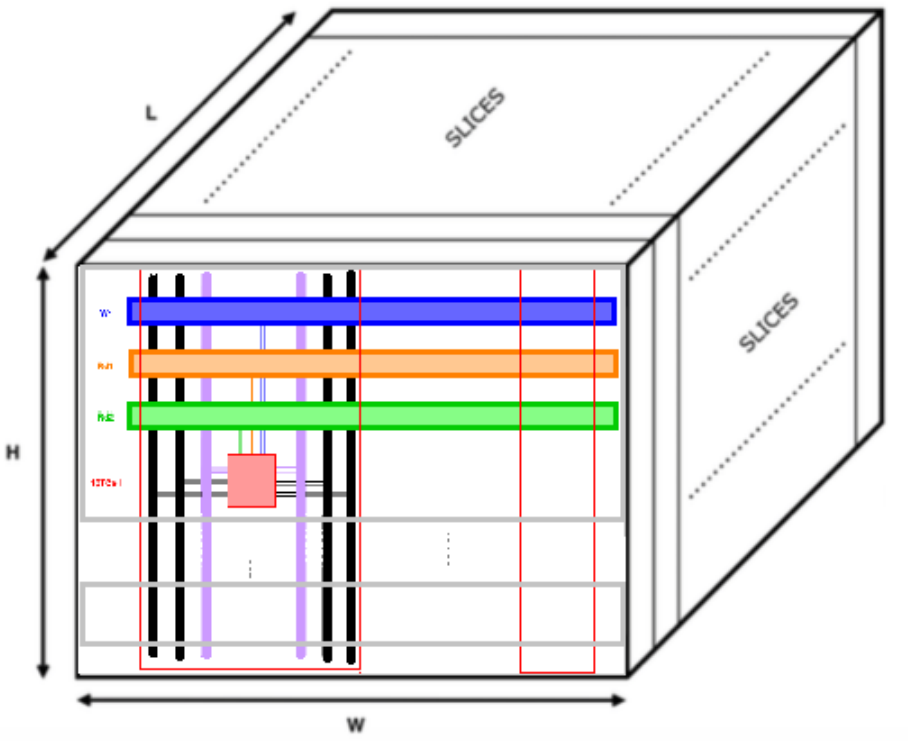

Figure 4.1: Simplified Register File structure

The three dimensions of the array have been computed as follows:

- *Width, Length*: In order to find these values, the area of the elementary bit cell has been calculated and then its square root has been used in order to compute width and length.

$$\mathbf{Area\_BitCell} = 2 \cdot (N\_port\_Wr + N\_port\_Rd) \cdot Tr\_n\_Area +$$
$$+2 \cdot (Tr\_n\_Area + Tr\_p\_Area)$$
$$\mathbf{W} = (2 \cdot (N\_port\_Wr + N\_port\_Rd) \cdot Pitch_{pp} + \sqrt{Area\_BitCell}) \cdot N_{bit} +$$
$$+Pitch_{pp} \cdot (N_{bit} - 1)$$
$$\mathbf{L} = N_{block} \cdot Pitch_{pp}$$

  As it can be seen, the area of the SRAM memory cell is computed as function of the number of read and write ports. Other elements, influenced by the number of input ports, are the number of wordline and bitline: this will affect the width and, as it can be seen in the following step, the height of the memory block.

- *Height of the stack*: the terms considered are the height of the Cells that takes into account of the the number of Read and Write lines and their distances, the square root of the Area of bit cells, and the distance between them, as shown in 4.2.

$$\mathbf{H} = ((N\_port\_Wr + N\_port\_Rd) \cdot Pitch_{pp} + \sqrt{Area\_BitCell}) \cdot N_{bit} +$$
$$+Pitch_{pp} \cdot (N_{bit} - 1)$$

  Finally, the area and the volume of the Register File can be calculated as:

$$\mathbf{Memory\_Array\_Area} = H \cdot W$$
$$\mathbf{Memory\_Array\_Volume} = H \cdot W \cdot L$$

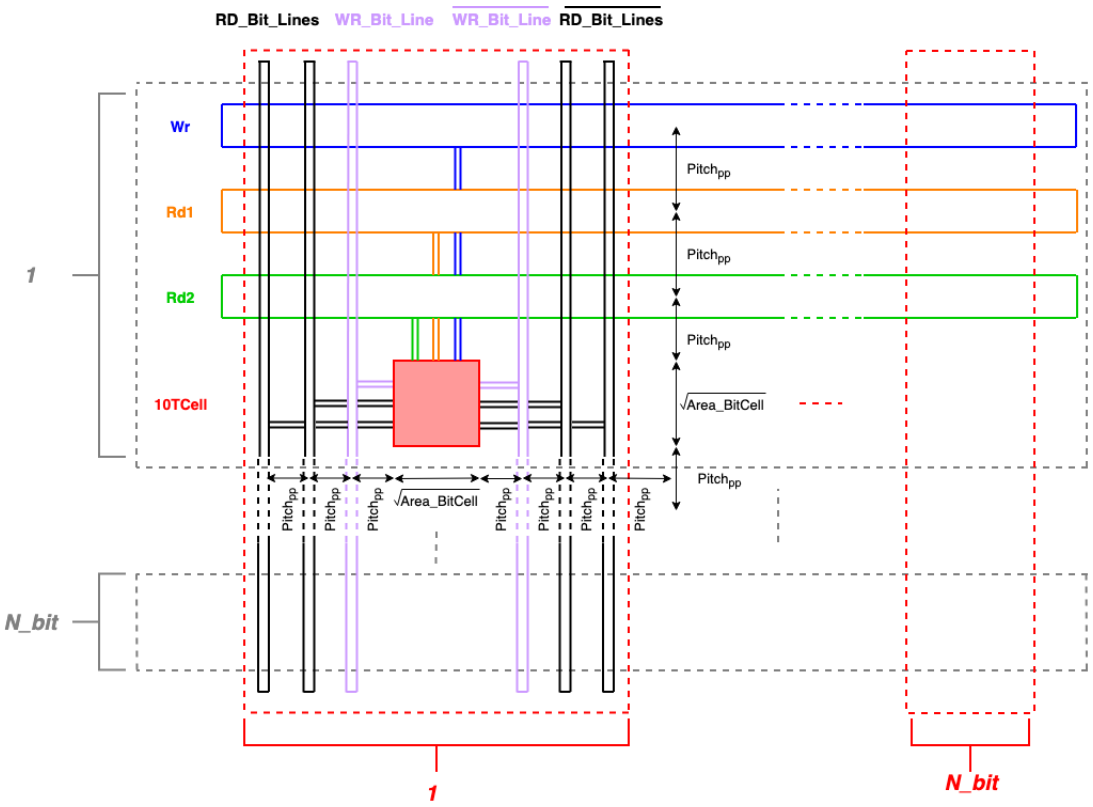

Figure 4.2: Stack structure

4.2 | Decoders

For the area and volume evaluation of all the decoders, the model that has been used is the same described in the Chapter 2. For each decoder the number of n-type transistors was first calculated and then the number of p-type transistors; subsequently they have been multiplied by their minimum area value and then added together to calculate the total area. Finally the volume has been calculated as the multiplication of the Area and the length.

- **Block Decoder**: The area of the *Block Decoder* has been calculated as: area of the the core of the decoder added to the area of the inverter connected at the input and and at the output. In this case the number of input is given by the *Block Address* while the output is equal to the number of slice N_{slice}. The volume is the area multiplied by the single distance between two transistor because there is only one *Block Decoder*. So:

$$\#Tr\_n\_Block\_Dec = Block\_Address \cdot N_{slice} + N_{slice} + Block\_Address$$

$$\#Tr\_p\_Block\_Dec = 2 \cdot N_{slice} + Block\_Address$$

$$\textbf{Block\_Dec\_Area} = \#Tr\_n\_Block\_Dec \cdot Tr\_n\_Area +$$
$$+\#Tr\_p\_Block\_Dec \cdot Tr\_p\_Area$$
$$\textbf{Block\_Dec\_Volume} = Block\_Dec\_Area \cdot Pitch_{pp}$$

- **Row Decoder**: The area of the *Row Decoder* has been calculated as before but in this case the number of input is done by the *Row Address* while the output is equal to the number of row that is (N_{wl}). This is finally multiplied by the total number of read and write ports. The volume is the area multiplied by the length. So:

$$\#Tr\_n\_Row\_Dec = Row\_Address \cdot N_{wl} + N_{wl} + Row\_Address$$

$$\#Tr\_p\_Row\_Dec = 2 \cdot N_{wl} + Row\_Address$$

$$\textbf{Row\_Dec\_Area} = (N\_port\_Wr + N\_port\_Rd) \cdot (\#Tr\_n\_Row\_Dec \cdot Tr\_n\_Area +$$
$$+\#Tr\_p\_Row\_Dec \cdot Tr\_p\_Area)$$
$$\textbf{Row\_Dec\_Volume} = Row\_Dec\_Area \cdot L$$

- **Decoder Address**: The previous variables $Block\_Address$, $Row\_Address$, N_wl and N_block are computed as:

$$Block\_Address = ceil(log_2(N_{block}))$$
$$Row\_Address = ceil(log_2(N_{wl}))$$
$$N\_wl = min(N\_word, N\_bit)$$
$$N\_block = ceil\Big(\frac{N\_word}{N\_bit}\Big)$$

4.3 | Bit Line Inverters, Pass Transistors and Precharge Transistors

In the total structure there are some pass transistor of n-type used to help the selection of the wanted memory cell.

- **Bit Line Inverters**: The area has been calculated as N_{bl} multiplied by the area of single p-type and n-type transistor p-type and n-type and then multiplied by the total number of read and write ports, while the volume is the area multiplied by the length. So:

$$\textbf{Inverter\_bl\_Area} = (N\_port\_Wr + N\_port\_Rd) \cdot \cdot (N_{bl} \cdot Tr\_n\_Area +$$
$$+ N_{bl} \cdot Tr\_p\_Area)$$
$$\textbf{Inverter\_bl\_Volume} = Inverter\_bl\_Area \cdot L$$

- **Row Pass Transistors**: The area has been calculated as N_{wl} multiplied by the area of single transistor and then multiplied by the total number of read and write ports, while the volume is the area multiplied by the length. So:

$$\textbf{Pass\_Row\_Area} = (N\_port\_Wr + N\_port\_Rd) \cdot N_{wl} \cdot Tr\_n\_Area$$
$$\textbf{Pass\_Row\_Volume} = Pass\_Row\_Area \cdot L$$

- **Column Pass Transistors**: The area has been calculated as N_{bl} multiplied by the area of single transistor and then multiplied by the total number of read ports, while the volume is the area multiplied by the length. So:

$$\textbf{Pass\_Column\_Area} = N\_port\_Rd \cdot N_{bl} \cdot Tr\_n\_Area$$
$$\textbf{Pass\_Column\_Volume} = Pass\_Column\_Area \cdot L$$

- **Slice Pass Transistors**: The area is the area of single transistor and then multiplied by the total number of read ports, while the volume is the area multiplied by the length. So:

$$\textbf{Pass\_Slice\_Area} = N\_port\_Rd \cdot Tr\_n\_Area$$

$$\textbf{Pass\_Slice\_Volume} = Pass\_Slice\_Area \cdot L$$

- **Precharge Transistors**: The area has been calculated as N_{bl} multiplied by the area of single p-type transistor and then multiplied by the total number of read ports, while the volume is the area multiplied by the length. So:

$$\textbf{Precharge\_Area} = N\_port\_Rd \cdot N_{bl} \cdot Tr\_p\_Area$$

$$\textbf{Precharge\_Volume} = Precharge\_Area \cdot L$$

4.4 | Sense Amplifier

The area of the two *Sense Amplifiers* has been calculated as remembering the structure described in Chapter 2 and the volume is the area multiplied by the length. So:

$$\#Tr\_n\_SA = N_{bl} \cdot 3$$
$$\#Tr\_p\_SA = N_{bl} \cdot 3$$

$$\textbf{SA\_Area} = N\_port\_Rd \cdot (\#Tr\_n\_SA \cdot Tr\_n\_Area + \#Tr\_p\_SA \cdot Tr\_p\_Area)$$
$$\textbf{SA\_Volume} = SA\_Area \cdot L$$

4.5 | Total Area and Total Volume

The total area has been calculated has the sum of the area of the memory array and all the boundary circuits:

$$\textbf{Total\_Area} = Memory\_Array\_Area + Block\_Dec\_Area + Row\_Dec\_Area +$$
$$+ Pass\_Row\_Area + Pass\_Column\_Area + Pass\_Slice\_Area +$$
$$+ Inverter\_bl\_Area + SA\_Area$$

The total volume has been calculated has the sum of the volume of the memory array and all the boundary circuits:

$$\textbf{Total\_Volume} = Memory\_Array\_Volume + Block\_Dec\_Volume + Row\_Dec\_Volume +$$
$$+ Pass\_Row\_Volume + Pass\_Column\_Volume + Pass\_Slice\_Volume +$$
$$+ Inverter\_bl\_Volume + SA\_Volume$$

4.6 | Simulation result

In order to verify the the computations made before, it has been made a simulation; in particular it has been reported two graphs in which are represented how the area and the volume change with the only value of N_{bit} since the assumption is that the Register File is a perfect cube. In particular, the array of values it has been considered for N_{bit} is $[8, 16, 32, 64, 128]$.
The behaviours represented are reasonable: in the two graphs 4.3 and 4.4 there is a square dependency. Changing the value of this parameter, the total area increases quadratically and the volume increase quadratically too.
The obtained results are reported in the following figures.

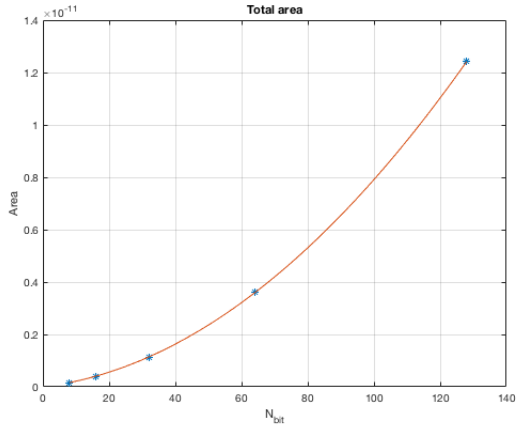

Figure 4.3: Simulation of the Total Area value, varying the number of bit.

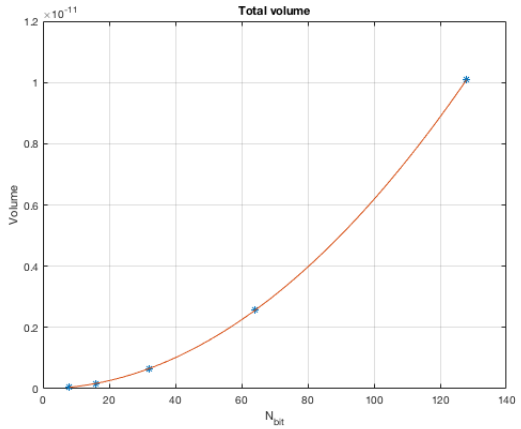

Figure 4.4: Simulation of the Total Volume value, varying the number of bit.

5 Matlab code

5.1 | Parameters.m

```
clear all
clc

%TEST PARAMETERS
%channel length
L = 1.00E-07;
%transistor width
W = 1.00E-06;
%ratio of mobilities pmos/nmos
Beta = 2;

%FILE PARAMETERS
%output capacitance of the driver driving the gates of the pmos inside the precharge unit
        C_ext_pu_driver = 20E-16*L;
%gate capacitante of a pmos inside the precharge unit
        Cg_pre = Beta*5E-16*L;
%gate capacitance of the pmos equalizer inside the precharge unit
        Cg_equalizer = Beta*5E-16*L;
%output resistance of the driver driving the gates of the pmos inside the precharge unit
        R_ext_pu_driver = 50;
%equivalent resistance of the precharge pmos driving the bitline
        Req_pre_p = 200;
%source capacitance of the precharge pmos driving the bitline
        Cs_pre = Beta*1.7E-16*L;
%interconnect resistance per unit of length of the bitline
        BL_r = 1.00E+07;
%interconnect capacitance per unit of length of the bitline
```

```
28          BL_c = 3.06E-11;
29  %interconnect capacitance per unit of length of the wordline
30      WL_c = 3.06E-11;
31  %drain capacitance of the sense amplifier p-MOS transistor
32          Cd_sa_p = Beta*1.7E-16*L;
33  %drain capacitance of the sense amplifier n-MOS transistor
34          Cd_sa_n = 1.7E-16*L;
35  %gate capacitance of the sense amplifier p-MOS transistor
36          Cg_sa_p = Beta*5E-16*L;
37  %gate capacitance of the sense amplifier n-MOS transistor
38          Cg_sa_n = 5E-16*L;
39  %drain capacitance of the evaluation n-mos transistor in the block decoder
40          Cd_bdec_n = 1.7E-16*L;
41  %drain capacitance of the evaluation n-mos transistor in the row decoder
42          Cd_rdec_n = 1.7E-16*L;
43  %gate capacitance of a nmos inside the block decoder
44          Cg_bdec_n = 5E-16*L;
45  %gate capacitance of a nmos inside the row decoder
46          Cg_rdec_n = 5E-16*L;
47  %drain capacitance of the pass transistor at the end of the bitline and connecting it
    ↪  to the external
48          Cd_blockpass = 1.7E-16*L;
49  %equivalent resistance of a nmos in the block decoder
50          Req_bdec_n = 200;
51  %drain capacitance of the precharge pmos inside the dynamic block decoder
52          Cd_bdec_pcharge = Beta*1.7E-16*L;
53  %gate capacitance of the pmos inside the inverter on the output of the block
    ↪  decoder
54          Cg_bdec_inv_p = Beta*5E-16*L;
55  %gate capacitance of the nmos inside the inverter on the output of the block
    ↪  decoder
56          Cg_bdec_inv_n = 5E-16*L;
57  %equivalent resistance of the pmos inside the inverter on the output of the block
    ↪  decoder
58          Req_bdec_inv_p = 200;
59  %drain resistance of the pmos inside the inverter on the output of the block
    ↪  decoder
60          Cd_bdec_inv_p = Beta*1.7E-16*L;
61  %drain resistance of the nmos inside the inverter on the output of the block
    ↪  decoder
62          Cd_bdec_inv_n = 1.7E-16*L;
63  %gate capacitance of a row pass transistor, on the output of the row decoder and
    ↪  connecting it to the wordline
```

```
64          Cg_rowpass = 5E-16*L;
65  %gate capacitance of the pass transistor at the end of the bitline and connecting it
    ↪  to the external
66          Cg_blockpass = 5E-16*L;
67  %equivalent resistance of a nmos in the row decoder
68          Req_rdec_n = 200;
69  %drain capacitance of the precharge pmos inside the dynamic row decoder
70          Cd_rdec_pcharge = Beta*1.7E-16*L;
71  %gate capacitance of the pmos inside the inverter on the output of the row
    ↪  decoder
72          Cg_rdec_inv_p = Beta*5E-16*L;
73  %gate capacitance of the nmos inside the inverter on the output of the row
    ↪  decoder
74          Cg_rdec_inv_n = 5E-16*L;
75  %equivalent resistance of the pmos inside the inverter on the output of the row
    ↪  decoder
76          Req_rdec_inv_p = 200;
77  %drain capacitance of the p_MOS of the inverter on the output of the row
    ↪  decoder
78          Cd_rdec_inv_p = Beta*1.7E-16*L;
79  %drain capacitance of the n_MOS of the inverter on the output of the row
    ↪  decoder
80          Cd_rdec_inv_n = 1.7E-16*L;
81  %equivalent resistance of a row pass transistor, on the output of the row decoder and
    ↪  connecting it to the wordline
82          Req_rowpass = 200;
83  %drain capacitance of a row pass transistor, on the output of the row decoder and
    ↪  connecting it to the wordline
84          Cd_rowpass = 1.7E-16*L;
85  %equivalent resistance of the pull-down nmos inside the memory cell
86          Req_cell_n = 200;
87  %equivalent resistance of the memory cell access nmos transistor
88          Req_access_n = 200;
89  %gate capacitance of the memory cell access nmos transistor
90          Cg_access = 5E-16*L;
91  %drain capacitance of the memory cell access nmos transistor
92          Cd_access = 1.7E-16*L;
93  %source capacitance of the memory cell access nmos transistor
94          Cs_access = 1.7E-16*L;
95  %percentage of voltage swing that makes the sense amplifier switch
96          K_SA = 0.05;
97  %input sense amplifier resistance
98          Req_sa_mod_parallel = 100;
```

```
99  %equivalent resistance of a column pass transistor, at the end of the bitline and
    ↪  connecting it to the external
100         Req_colpass = 200;
101 %equivalent resistance of the external driver forcing the value to be written on the
    ↪  bitlines
102         Req_driver = 25;
103 %output capacitance of the external driver forcing the value to be written on the
    ↪  bitlines
104         C_driver = 40E-16*L;
105 %equivalent resistance of the pull-up pmos inside the memory cell
106         Req_cell_p = 200;
107 %gate capacitance of the pull-up pmos inside the memory cell
108         Cg_cell_p = Beta*5E-16*L;
109 %gate capacitance of the pull-down nmos inside the memory cell
110         Cg_cell_n = 5E-16*L;
111 %drain capacitance of the pull-up pmos inside the memory cell
112         Cd_cell_p = Beta*1.7E-16*L;
113 %drain capacitance of the pull-down nmos inside the memory cell
114         Cd_cell_n = 1.7E-16*L;
115 %number of words in the memory
116         N_word = 128;
117 %number of bits in each word
118         N_bit_array = [8, 16, 32, 64, 128];
119 %precharge voltage
120         V_bl_prec = 1.2;
121 %pass-transistor voltage
122         V_on_pt = 1.2;
123 %selected wordline voltage
124         V_sel = 1.2;
125 %unselected wordline voltage
126         V_unsel = 0;
127 %zero read voltage
128         V_rd_0 = 0.8;
129 %one read voltage
130         V_rd_1 = 1.2;
131 %programming bitline voltage
132         V_prog = 1.2;
133 %unprogramming bitline voltage
134         V_unprog = 0;
135 %Read frequency
136     f_read = 5E6;
137 %Write frequency
138     f_write = 5E6;
```

```
%number of write ports
        N_port_Wr = 1;
%number of read ports
        N_port_Rd = 2;

        Tr_n_Area = W*L;

        Tr_p_Area = Beta*Tr_n_Area;
%distance between columns of transistor
        Pitch_pp = 3.00E-07;



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%COMPUTATIONS

for i=1:numel(N_bit_array)
    N_bit = N_bit_array(i);

        [ total_read_power, total_write_power, Total_delay_read,  Total_delay_write]
        ↪  = Register_File (C_ext_pu_driver, ...
        Cg_pre, ...
        Cg_equalizer, ...
        R_ext_pu_driver, ...
        Req_pre_p, ...
        Cs_pre, ...
        BL_r, ...
        BL_c, ...
    WL_c, ...
        Cd_sa_p, ...
        Cd_sa_n, ...
        Cg_sa_p, ...
        Cg_sa_n, ...
        Cd_bdec_n, ...
        Cd_rdec_n, ...
        Cg_bdec_n, ...
        Cg_rdec_n, ...
        Cd_blockpass, ...
        Req_bdec_n, ...
        Cd_bdec_pcharge, ...
        Cg_bdec_inv_p, ...
```

```
181         Cg_bdec_inv_n, ...
182         Req_bdec_inv_p, ...
183         Cd_bdec_inv_p, ...
184         Cd_bdec_inv_n, ...
185         Cg_rowpass, ...
186         Cg_blockpass, ...
187         Req_rdec_n, ...
188         Cd_rdec_pcharge, ...
189         Cg_rdec_inv_p, ...
190         Cg_rdec_inv_n, ...
191         Req_rdec_inv_p, ...
192         Cd_rdec_inv_p, ...
193         Cd_rdec_inv_n, ...
194         Req_rowpass, ...
195         Cd_rowpass, ...
196         Req_cell_n, ...
197         Req_access_n, ...
198         Cg_access, ...
199         Cd_access, ...
200         Cs_access, ...
201         K_SA, ...
202         Req_sa_mod_parallel, ...
203         Req_colpass, ...
204         Req_driver, ...
205         C_driver, ...
206         Req_cell_p, ...
207         Cg_cell_p, ...
208         Cg_cell_n, ...
209         Cd_cell_p, ...
210         Cd_cell_n, ...
211         N_word, ...
212         N_bit,...
213     V_bl_prec,...
214         V_on_pt,...
215         V_sel,...
216         V_unsel,...
217         V_rd_0,...
218         V_rd_1,...
219         V_prog,...
220         V_unprog,...
221     f_read,...
222     f_write,...
223         N_port_Wr,...
```

```
224         N_port_Rd,...
225         Tr_n_Area,...
226         Tr_p_Area,...
227         Pitch_pp);
228 
229         total_read_power_array(i) = total_read_power;
230         total_write_power_array(i) =  total_write_power;
231         Total_delay_read_array(i) = Total_delay_read;
232         Total_delay_write_array(i) = Total_delay_write;
233     Total_area_array(i) = Total_delay_read;
234         Total_volume_array(i) = Total_delay_write;
235 
236 end
237 figure(1)
238 xq= 8:1:128;
239 s = spline(N_bit_array,Total_delay_read_array,xq); %xq coordinate x punti
    ↪  interpolazione, s coordinate y punti interpolazione
240 plot(N_bit_array,Total_delay_read_array, '*', xq, s)
241 title('Total delay - read')
242 xlabel('N_{bit}')
243 ylabel('Delay')
244 grid on
245 print('delay_read','-depsc')
246 
247 figure(2)
248 xq= 8:1:128;
249 s = spline(N_bit_array,Total_delay_write_array,xq); %xq coordinate x punti
    ↪  interpolazione, s coordinate y punti interpolazione
250 plot(N_bit_array,Total_delay_write_array, '*', xq, s)
251 title('Total delay - write')
252 xlabel('N_{bit}')
253 ylabel('Delay')
254 grid on
255 print('delay_write','-depsc')
256 
257 figure(3)
258 xq= 8:1:128;
259 s = spline(N_bit_array,Total_area_array,xq); %xq coordinate x punti interpolazione, s
    ↪  coordinate y punti interpolazione
260 plot(N_bit_array,Total_area_array, '*', xq, s)
261 title('Total area')
262 xlabel('N_{bit}')
263 ylabel('Area')
```

```
grid on

figure(4)
xq= 8:1:128;
s = spline(N_bit_array,Total_volume_array,xq); %xq coordinate x punti interpolazione, s coordinate y punti interpolazione
plot(N_bit_array,Total_volume_array, '*', xq, s)
title('Total volume')
xlabel('N_{bit}')
ylabel('Volume')
grid on

figure(5)
xq= 8:1:128;
s = spline(N_bit_array,total_read_power_array,xq); %xq coordinate x punti interpolazione, s coordinate y punti interpolazione
plot(N_bit_array,total_read_power_array, '*', xq, s)
title('Read Power')
xlabel('N_{bit}')
ylabel('Power')
grid on

figure(6)
xq= 8:1:128;
s = spline(N_bit_array,total_write_power_array,xq); %xq coordinate x punti interpolazione, s coordinate y punti interpolazione
plot(N_bit_array,total_write_power_array, '*', xq, s)
title('Write Power')
xlabel('N_{bit}')
ylabel('Power')
grid on
```

5.2 | Register_File.m

```
function [ total_read_power, total_write_power, Total_delay_read, Total_delay_write,Total_area, Total_volume] = Register_File (C_ext_pu_driver, ...
Cg_pre, ...
Cg_equalizer, ...
R_ext_pu_driver, ...
```

```
Req_pre_p, ...
Cs_pre, ...
BL_r, ...
BL_c, ...
WL_c, ...
Cd_sa_p, ...
Cd_sa_n, ...
Cg_sa_p, ...
Cg_sa_n, ...
Cd_bdec_n, ...
Cd_rdec_n, ...
Cg_bdec_n, ...
Cg_rdec_n, ...
Cd_blockpass, ...
Req_bdec_n, ...
Cd_bdec_pcharge, ...
Cg_bdec_inv_p, ...
Cg_bdec_inv_n, ...
Req_bdec_inv_p, ...
Cd_bdec_inv_p, ...
Cd_bdec_inv_n, ...
Cg_rowpass, ...
Cg_blockpass, ...
Req_rdec_n, ...
Cd_rdec_pcharge, ...
Cg_rdec_inv_p, ...
Cg_rdec_inv_n, ...
Req_rdec_inv_p, ...
Cd_rdec_inv_p, ...
Cd_rdec_inv_n, ...
Req_rowpass, ...
Cd_rowpass, ...
Req_cell_n, ...
Req_access_n, ...
Cg_access, ...
Cd_access, ...
Cs_access, ...
K_SA, ...
Req_sa_mod_parallel, ...
Req_colpass, ...
Req_driver, ...
C_driver, ...
Req_cell_p, ...
```

```
Cg_cell_p, ...
Cg_cell_n, ...
Cd_cell_p, ...
Cd_cell_n, ...
N_word, ...
N_bit,...
V_bl_prec,...
V_on_pt,...
V_sel,...
V_unsel,...
V_rd_0,...
V_rd_1,...
V_prog,...
V_unprog,...
f_read,...
f_write,...
N_port_Wr,...
N_port_Rd,...
Tr_n_Area,...
Tr_p_Area,...
Pitch_pp)

%% MEMORY PARAMETERS - COMPUTED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

N_block = ceil(N_word/N_bit);
N_wl=min(N_word,N_bit);
Block_Address= ceil(log2(N_block));
Row_Address= ceil(log2(N_wl));


%% GEOMETRY PARAMETERS - COMPUTED
% width of the array of memory
        Area_BitCell=2*(N_port_Wr+N_port_Rd)*Tr_n_Area+2*(Tr_n_Area+Tr_p_Area);
        W=(2*(N_port_Wr+N_port_Rd)*Pitch_pp+sqrt(Area_BitCell))*N_bit+Pitch_pp*(N_bit-1);
% length of the array of memory
        L=N_block*Pitch_pp; %ok
% height of the array of memory
        H=((N_port_Wr+N_port_Rd)*Pitch_pp+sqrt(Area_BitCell))*N_bit+Pitch_pp*(N_bit-1);
% length of bitline
    BL_length=H;
% length of wordline
        WL_length=W;

```

```
91  %% COMMON PARAMETERS - COMPUTED
92          Cd_bdec_eval=Block_Address*Cd_bdec_n;
93          Cd_rdec_eval=Row_Address*Cd_rdec_n;
94
95
96
97  %Computed Capacitances
98  Cd_access_l=(Cd_access*N_wl)/BL_length;
99  Cg_access_l=2*(Cg_access*N_bit)/WL_length;
100 C_prec=C_ext_pu_driver+2*Cg_pre+Cg_equalizer;
101 C_SA_in=Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n;
102 C_bl=Cs_pre+BL_length*BL_c+Cd_access*N_wl+C_SA_in;
103 C_bl_write=C_ext_pu_driver+BL_length*BL_c+Cd_access*N_wl;
104 C_wl=Cd_rowpass+WL_c*WL_length+2*Cg_access*N_bit;
105 C_block_stack=Cg_bdec_n*Block_Address*N_block;
106 Cd_block_dec=Cd_bdec_pcharge+Cd_bdec_eval+Cg_bdec_inv_p+Cg_bdec_inv_n;
107 C_row_stack=Cg_rdec_n*Row_Address*N_wl;
108 Cd_row_dec=Cd_rdec_pcharge+Cd_rdec_eval+Cg_rdec_inv_p+Cg_rdec_inv_n;
109
110 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
111 %READ ENERGY
112 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
113 total_read_energy=0;
114
115 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
116 %Precharge path
117 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
118
119 %Precharge unit energy
120         total_read_energy=0.5*C_prec*((V_on_pt)^2)*N_bit+total_read_energy;
121
122 %Bit line charge energy
123         total_read_energy=0.5*C_bl*(2*(V_bl_prec)^2)*N_bit+total_read_energy;  %for
            ↪  both bit lines
124
125 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
126 %Cell selecting path (DECODERS)
127 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
128
129 %Block decoder evaluation stack charge energy
130         total_read_energy=0.5*C_block_stack*(V_on_pt^2)+total_read_energy;
131
132 %Block decoder energy
```

```
133         total_read_energy=0.5*Cd_block_dec*(V_on_pt^2)+total_read_energy;
134
135 %Row decoder evaluation stack charge energy
136         total_read_energy=0.5*C_row_stack*(V_on_pt^2)*N_block+total_read_energy;
137
138 %Row decoder energy
139         total_read_energy=0.5*Cd_row_dec*(V_sel^2+(N_wl-1)*V_unsel^2)*N_block+total_read_energy;
140
141 %Selection block energy
142         total_read_energy=
            ↪  0.5*(Cg_rowpass*N_wl+Cg_blockpass*N_bit)*V_on_pt^2+total_read_energy;
143
144 %Selected and Unselected Word lines energy
145         total_read_energy=0.5*C_wl*(V_sel^2+(N_wl-1)*V_unsel^2)+total_read_energy;
146
147 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
148 %Sensing path
149 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
150
151 %Bit line discharge energy (sensing energy is included)
152         total_read_energy=0.5*C_bl*V_bl_prec*(abs(V_bl_prec-V_rd_0)+abs(V_bl_prec-V_rd_1))*N_bit+tot
153
154 % energy of the sense amplifier
155         total_read_energy=
            ↪  0.5*Cd_blockpass*V_bl_prec*(abs(V_bl_prec-V_rd_0)+abs(V_bl_prec-V_rd_1))*N_bit+total_rea
156
157  total_read_power = total_read_energy*f_read;
158
159
160
161 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
162 %WRITE ENERGY
163 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
164
165 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
166 %Precharge path
167 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
168 total_write_energy=0;
169 %Load charge energy
170         total_write_energy=0.5*C_bl_write*((V_prog)^2+(V_unprog)^2)*N_bit+total_write_energy;
171
172 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
173 %Cell selecting path (DECODERS)
```

```
174 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
175
176 %Block decoder evaluation stack charge energy
177         total_write_energy=0.5*C_block_stack*(V_on_pt^2)+total_write_energy;
178
179 %Block decoder energy
180         total_write_energy=0.5*Cd_block_dec*(V_on_pt^2)+total_write_energy;
181
182 %Row decoder evaluation stack charge energy
183         total_write_energy=0.5*C_row_stack*(V_on_pt^2)*N_block+total_write_energy;
184
185 %Row decoder energy
186         total_write_energy=0.5*Cd_row_dec*(V_sel^2+(N_wl-1)*V_unsel^2)*N_block+total_write_energy;
187
188 %Selection block energy
189         total_write_energy= 0.5*(Cg_rowpass*N_wl)*(V_on_pt^2)+total_write_energy;
190
191 %Selected and Unselected Word lines energy
192         total_write_energy=0.5*C_wl*(V_sel^2+(N_wl-1)*V_unsel^2)+total_write_energy;
193
194  total_write_power = total_write_energy*f_write;
195
196 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
197 %% DELAY
198 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
199
200 %READ DELAY
201 %total delay: computed in a read operation (which includes also a precharge
202 %operation)
203 del=0;
204
205 %Delay to activate precharge unit - RC delay
206     del = del + R_ext_pu_driver*(C_ext_pu_driver+2*Cg_pre+Cg_equalizer);
207
208 %Precharge delay - Bakoglu delay
209         Rd=Req_pre_p;
210         Cd=Cs_pre;
211         r=BL_r;
212         c=BL_c+Cd_access_l;
213         l=BL_length;
214         Cl=(Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n)+Cd_blockpass;
215     del = del + ((Rd*(Cd+c*l+Cl)+r*l*Cl)+0.377/0.69*(r*c*l^2));
216
```

```
217 %Block decoder delay - Gate delay
218         Req_n=Req_bdec_n;
219         Cd=Cd_bdec_pcharge+Cd_bdec_eval;
220         Cl=Cg_bdec_inv_p+Cg_bdec_inv_n;
221         Stack_n=Block_Address;
222         R_n=Stack_n*Req_n;
223     del_block_dec=R_n*(Cd+Cl);
224
225 %Inverter on block decoder's output delay - Gate delay
226         Req_p=Req_bdec_inv_p;
227         Cd=Cd_bdec_inv_p+Cd_bdec_inv_n;
228         Cl=N_wl*Cg_rowpass+2*N_bit*Cg_blockpass;
229         Stack_p=1;
230         R_p=Stack_p*Req_p;
231         del_block_inv = R_p*(Cd+Cl);
232
233 %Row decoder delay - Gate delay
234         Req_n=Req_rdec_n;
235         Cd=Cd_rdec_pcharge+Cd_rdec_eval;
236         Cl=Cg_rdec_inv_p+Cg_rdec_inv_n;
237         Stack_n=Row_Address;
238         R_n=Stack_n*Req_n;
239     del_row_dec=R_n*(Cd+Cl);
240
241 %Word line charge delay - Elmore delay (l'inverter del decoder di riga lo considero
    ↪  qui)
242         Req_inv=Req_rdec_inv_p;
243         Cd_inv=Cd_rdec_inv_p+Cd_rdec_inv_n;
244         Req_pass=Req_rowpass;
245         Cd_pass=Cd_rowpass;
246         Cl=2*Cg_access_l*WL_length;
247         del_row_inv = (Req_inv*(Cd_inv+Cd_pass+Cl)+Req_pass*(Cd_pass+Cl));
248
249 %Bit line discharge delay (the 0.05 factor is the fraction of the bit line
250 %delay that influences the total delay, because afterwards the sense
251 %amplifier is triggered. About 5-10 mV should be the initial differential
252 %signal for the sense amplifier) - Distributed delay
253         Rd=Req_cell_n+Req_access_n;
254         Cd=Cd_access;
255         r=BL_r;
256         c=BL_c+Cd_access_l;
257         l=BL_length;
258         Cl=Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n+Cd_blockpass;
```

```
259         del= del + K_SA*((Rd*(Cd+c*l+Cl)+r*l*Cl)+0.377/0.69*(r*c*l^2));
260
261 %Sense amplifier delay
262     del= del +
        ↪  Req_sa_mod_parallel*(Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n+(BL_c*BL_length)+Cd_blockpass);
263
264 %Column pass transistor and block transistor delay (after switching of SA) - Elmore
    ↪  delay
265         Req_pass=Req_colpass;
266         Cd_pass=Cd_blockpass;
267         Cl=0;         %assumo condizione a vuoto
268         del = del + (Req_pass*(Cd_pass+Cl));
269
270 if(del_block_dec>del_row_dec)
271         del=del_block_dec+del;
272 else
273         del=del_row_dec+del;
274 end
275
276 if(del_block_inv>del_row_inv)
277         del=del_block_inv+del;
278 else
279         del=del_row_inv+del;
280 end
281
282 Total_delay_read=0.69*del;
283
284 %WRITE DELAY
285 %Block decoder delay - Gate delay
286         Req_n=Req_bdec_n;
287         Cd=Cd_bdec_pcharge+Cd_bdec_eval;
288         Cl=Cg_bdec_inv_p+Cg_bdec_inv_n;
289         Stack_n=Block_Address;
290         R_n=Stack_n*Req_n;
291     del_block_dec=R_n*(Cd+Cl);
292
293 %Inverter on block decoder's output delay - Gate delay
294         Req_p=Req_bdec_inv_p;
295         Cd=Cd_bdec_inv_p+Cd_bdec_inv_n;
296         Cl=N_wl*Cg_rowpass;
297         Stack_p=1;
298         R_p=Stack_p*Req_p;
299         del_block_inv = R_p*(Cd+Cl);
```

```
300
301 %Row decoder delay - Gate delay
302         Req_n=Req_rdec_n;
303         Cd=Cd_rdec_pcharge+Cd_rdec_eval;
304         Cl=Cg_rdec_inv_p+Cg_rdec_inv_n;
305         Stack_n=Row_Address;
306         R_n=Stack_n*Req_n;
307     del_row_dec=R_n*(Cd+Cl);
308
309 %Word line charge delay - Elmore delay (l'inverter del decoder di riga lo considero
    ↪  qui)
310         Req_inv=Req_rdec_inv_p;
311         Cd_inv=Cd_rdec_inv_p+Cd_rdec_inv_n;
312         Req_pass=Req_rowpass;
313         Cd_pass=Cd_rowpass;
314         Cl=2*Cg_access_l*WL_length;
315         del_row_inv = (Req_inv*(Cd_inv+Cd_pass+Cl)+Req_pass*(Cd_pass+Cl));
316
317 if(del_block_dec>del_row_dec)
318         del=del_block_dec;
319 else
320         del=del_row_dec;
321 end
322
323 if(del_block_inv>del_row_inv)
324         del=del_block_inv+del;
325 else
326         del=del_row_inv+del;
327 end
328
329 %Driver delay - Bakoglu delay
330         Rd=Req_driver;
331         Cd=C_driver;
332         r=BL_r;
333         c=BL_c+Cd_access_l;
334         l=BL_length;
335         Cl=0;
336     del = del + ((Rd*(Cd+c*l+Cl)+r*l*Cl)+0.377/0.69*(r*c*l^2));
337
338 %Cell delay
339 tau_1=1/(1/Req_cell_p+1/Req_access_n)*(Cg_cell_p+Cg_cell_n+Cd_cell_p+Cd_cell_n+Cs_access);
340 tau_2=Req_cell_p*(Cg_cell_p+Cg_cell_n+Cd_cell_p+Cd_cell_n+Cs_access);
341 del= del + 4*tau_1 + tau_2;
```

```
342
343 Total_delay_write=0.69*del;
344
345 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
346 %% AREA AND VOLUME
347 %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
348 % block decoder
349         n_Tr_n_Block_Dec = Block_Address*N_block+N_block+Block_Address;
350         n_Tr_p_Block_Dec = 2*N_block+Block_Address;
351         Block_Dec_Area = n_Tr_n_Block_Dec*Tr_n_Area + n_Tr_p_Block_Dec*Tr_p_Area;
352 % row decoder
353         n_Tr_n_Row_Dec =
            ↪  Row_Address*N_wl+N_wl+Row_Address;
354         n_Tr_p_Row_Dec = 2*N_wl+ Row_Address;
355         Row_Dec_Area = (N_port_Wr+N_port_Rd)*(n_Tr_n_Row_Dec*Tr_n_Area +
            ↪  n_Tr_p_Row_Dec*Tr_p_Area);
356 % inverter bitline
357         Inv_bl_Area = (N_port_Wr+N_port_Rd)*(N_bit*Tr_n_Area +
            ↪  N_bit*Tr_p_Area);
358 % pass_block_row
359         n_Tr_n_pass_row_Area =
            ↪  (N_port_Wr+N_port_Rd)*N_wl*Tr_n_Area;
360 % pass_block_block
361         n_Tr_n_pass_block_Area =
            ↪  N_block*N_port_Rd*Tr_n_Area;
362 % precharge_p_mos
363         n_Tr_p_precharge_Area =
            ↪  N_port_Rd*N_bit*(3*Tr_p_Area);
364 % sense amplifier
365         n_Tr_n_SA = N_bit*3;
366         n_Tr_p_SA = N_bit*3;
367         SA_Area = N_port_Rd*(n_Tr_n_SA*Tr_n_Area +
            ↪  n_Tr_p_SA*Tr_p_Area);
368
369 %Boundary component area (block decoder, row decoder, sense amplifier, pass
    ↪  transistors, precharge_p_mos)
370         Area_bound_comp =         Block_Dec_Area + Row_Dec_Area + SA_Area +...
371                                                 n_Tr_n_pass_row_Area  +
                                                    ↪  n_Tr_n_pass_block_Area +...
372                                                 n_Tr_p_precharge_Area + Inv_bl_Area;
                                                    ↪
373 %Boundary component Volume (block decoder, row decoder, colum decoder, sense
    ↪  amplifier, pass transistors, precharge_p_mos)
```

```
374         Volume_bound_comp = Area_bound_comp*L;                %first output from the
            ↪ function
375
376 % array area
377         Area_memory=H*W;
378 % array volume
379         Volume_memory=Area_memory*L;
            ↪                                              %Second output from the function
380
381 % total area (total number of transistor * area of single transistor)
382     Total_area = Area_memory + Area_bound_comp;
383 % total volume (sum of volumes memory and boundary component)
384         Total_volume = Volume_memory + Volume_bound_comp;                %Third
            ↪ output from the function
385
386 end
```