Contents

List of Figures

1 Introduction

1.1 | Introduction to the laboratory experience

This project aims to estimate the characteristics of a 3D NAND memory array based on Traditional Planar Transistors through a MATLAB script.

The outputs of the model are:

- Delay during a precharge and read operation
- Power consumption due to a read operation
- Power consumption due to a write operation
- Power consumption due to an erase operation
- Area and Volume

1.2 | Structure explanation

The 3D NAND memory exploits the vertical dimension to increase the bit density of the array. From a logical point of view, the model is represented by a series of memory blocks called "slices" that together form the array, as shown in fig. 1.1. Every slice works like a traditional planar NAND memory block. So, after selecting a slice, the behaviour of the memory is the same as a planar one. All around the memory slices there are some pieces of circuit (Decoders) used to select the desired memory cell, plus other circuits, like Sense Amplifiers and Precharge Units, as shown in fig. 1.2.

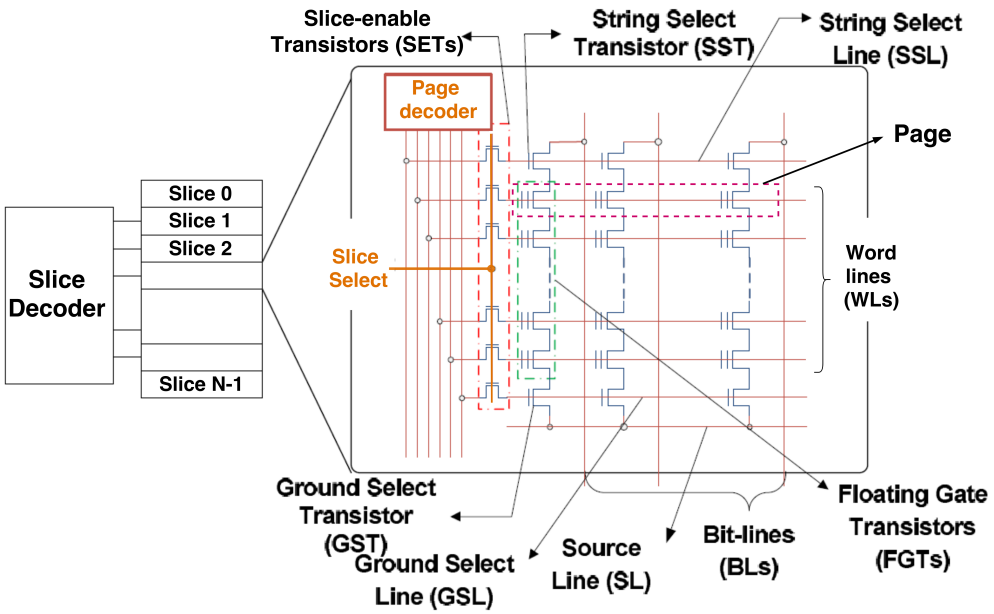

Figure 1.1: Block structure of the memory

The elements that dissipate energy are:

- String select line (SSL)
- Ground select line (GSL)
- Bitlines (BLs)
- Wordlines (WLs)
- Slice enable transistors (SETs)
- String select transistor (SST)
- Ground select transistor (GST)
- Floating gate transistors (FGTs)

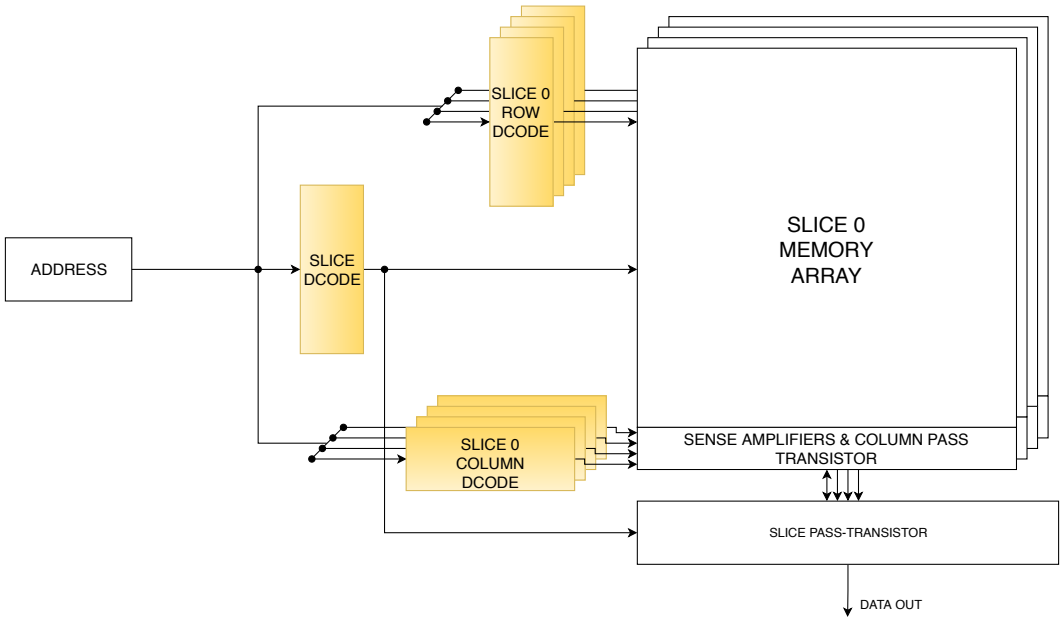

Figure 1.2: Block diagram of the total structure

1.3 | Behavior of the memory

The behaviour of the memory can be modelled like a FSM with these states, as shown in fig. 1.3:

- Precharge
- Read
- Write
- Erase

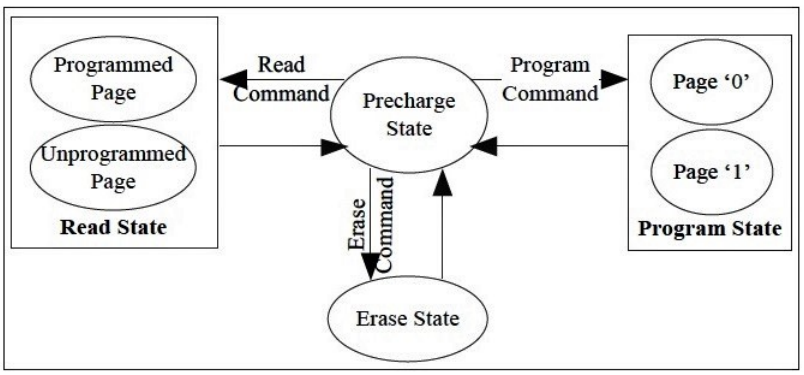

Figure 1.3: State machine for memory operation

Precharge is the initial state to perform a **read**, **write** or **erase** operation. During the precharge state, bitlines are biased at a certain voltage to speed up subsequent operations. Bitlines are electrically disconnected from the memory array because the SSL signal is not asserted. Thanks to slice enable transistors (SETs), the wordlines are electrically disconnected too.

Read and **write** operations are performed at page level while erase at slice level. During any operation, unselected slices are electrically disconnected by the slice decoder.

The memory array in standard planar NAND is obviously a plane but in 3D ones it has a cubic shape. The cube is organized in blocks that in the model are referred to as *slices*. A slice decoder is used to select only the desired slice and to electrically isolate the others.

The layout is organized in rows (*wordlines*) and columns (*bitlines*). At the intersection of each row and column there is a Floating Gate Transistor (FGT) that is the actual memory element. In the project a SLC (single-level cell) flash is used so each FGT stores only one bit of data. The FGTs connected in series form a *string* and can be accessed using the String Select Transistor (SST) and the Ground Select Transistor (GST). The group of FGTs along the same WL is called *page* and it is selected using the page decoder.

Flash memory exploits Tunneling Effect to perform write/erase operations on the FGTs. The write operation moves the tunneling charges in the floating gate, while they are extracted from it during the erase phase.

The model of the memory incudes also all the peripheral components needed to select the bits and to read them: apart from the memory slices, so, we have modelled also the slice, the row and column decoders, the sense amplifiers and all the pass transistors needed for the correct selection of the lines and for providing the data towards the external world. The behaviour of each component during a read operation will be better detailed in chapter 2.

We report in fig. 1.4 a more detailed scheme of the memory. We must remember that, being a flash memory, the data to be stored inside the chip is typically huge; for this reason each row of each slice will for sure have to host more than a single word. For example, we could consider a slice whose dimension is 1024×1024 bits: if a single word has 64 bits of parallelism, in each row we'll have 16 words stored.

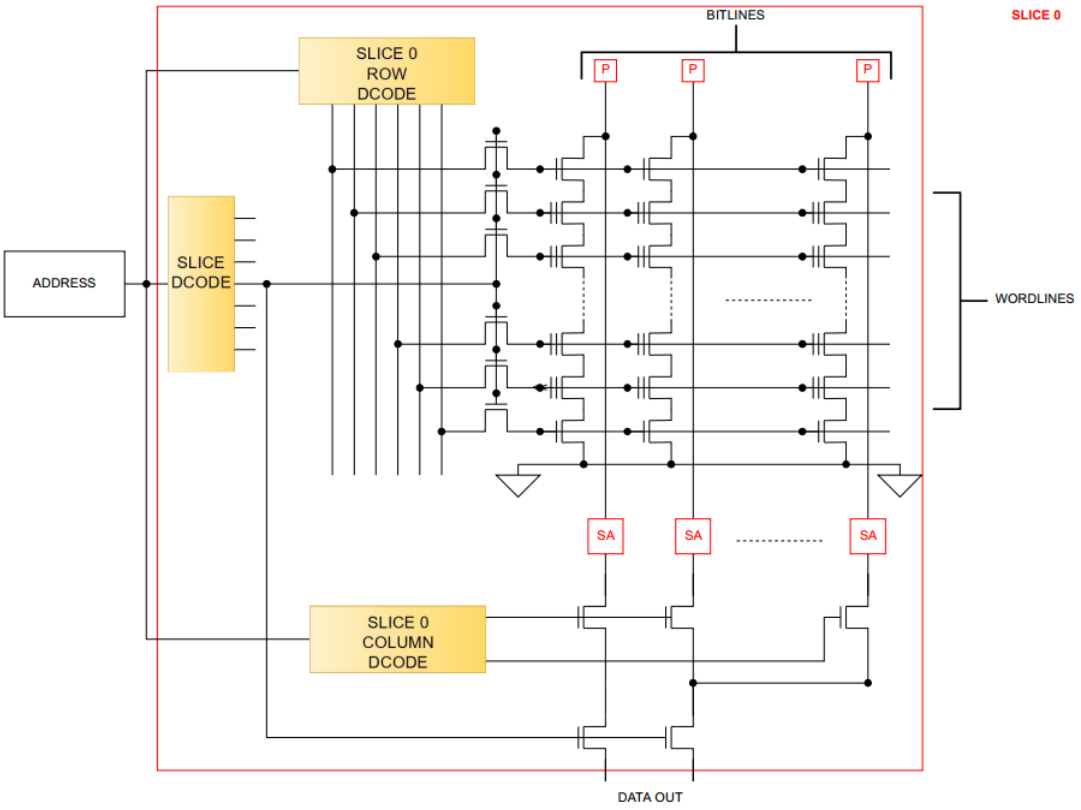

Figure 1.4: Detailed memory scheme

Let's say, to make a practical example which can be adapted to fig. 1.4, that instead of 64 bits of parallelism, each word has only 2 bits. Then, in an array 1024×1024, we would have 512 words per row. After the activation of the corresponding wordline, all these 512 words will force the value of their bits on the 1024 bitlines. So the 1024 sense amplifiers will accelerate the detection of the stored value. However, the column decoder will allow only the correct couple out of these 512 couples to reach the drain of the two last pass transistors, which transmit the correct 2-bits read word to the external world.

The parameters provided in input to the model, then, are N_{bl} (number of bitlines per slice), N_{wl} (number of wordlines per slice), N_{slice} (number of slices) and $N_{bit,word}$ (number of bits per word). The number of bits per address so are computed like:

$$Block\ Address = \lceil log_2(N_{slice}) \rceil$$

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

$$Column\ Address = \left\lceil log_2\left(\frac{N_{bl}}{N_{bit,word}}\right) \right\rceil$$

Delay

2

2.1 | Delay computation

Each operation on the memory requires a certain time to be performed. In the model used four operations have been defined: precharge, read, write and erase. The only delays that have been considered in the model are precharge and read ones, because they are the only ones that can be evaluated in a qualitative way by using the parameters defined previously. The write and erase delays depend on physical and technological parameters of the transistors that are involved in the tunnel effect, so they are hard to model. Another thing that has been considered is the fact that the most common operation on a NAND memory is the read operation (with its precharge). So, we estimate the delay due to a precharge and subsequent read operation, taking into account not only the memory array, but also all the hardware components around it, from the decoding of the address bits to the switching of the sense amplifier and the selection of the column bit to be read.

2.1.1 | 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). The precharge unit is simply a pmos transistor, connected from one side to the supply voltage and from the other side to the bitline. The *Enable* signal has to discharge the gate capacitance of this pmos, in order to make it able to switch, so the delay associated to this operation is:

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

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

$$\tau = \frac{(C_{bl,wire} \cdot L_{bl})V_{bl,prec}}{I_{on,driver}}$$

where L is the length of the array of memory.

2.1.2 | 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:

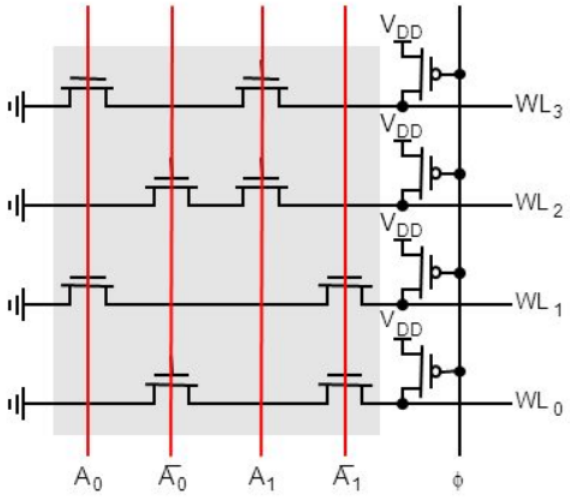

Figure 2.1: Core of the dynamic NAND decoder

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.

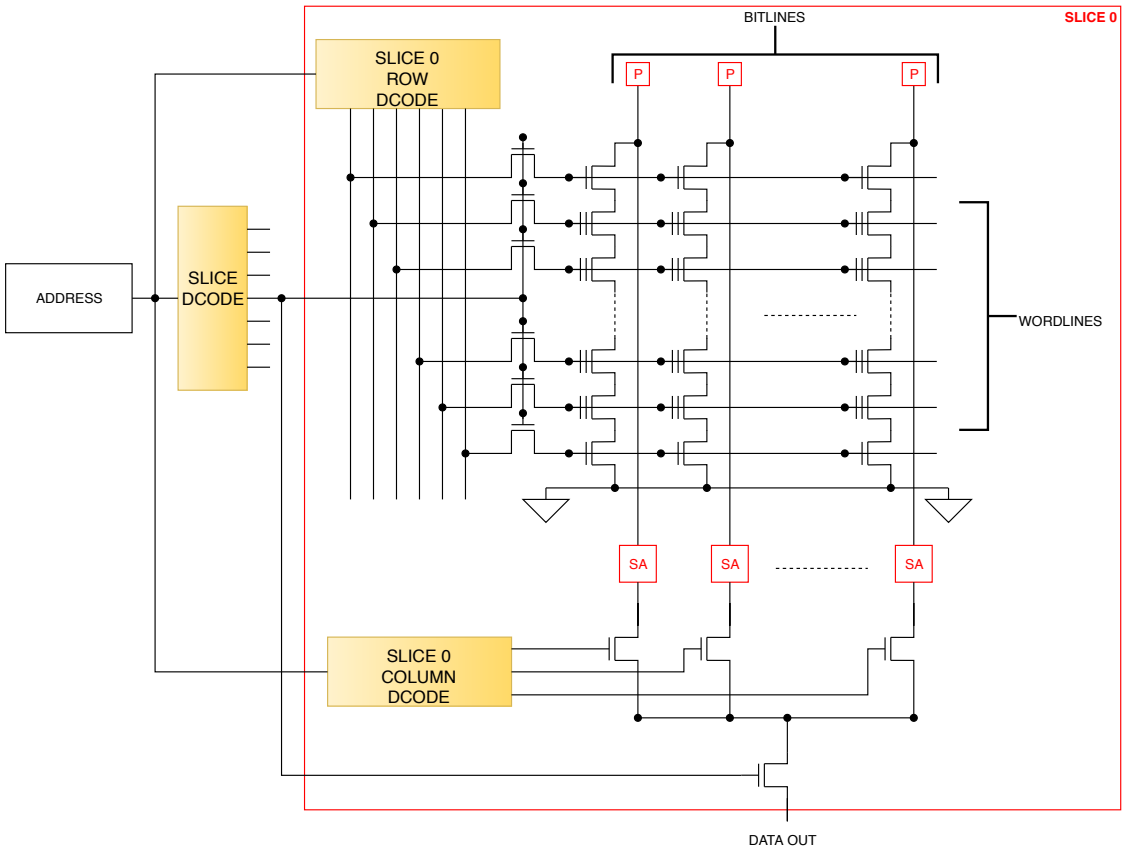

Figure 2.2: Full scheme of the memory

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:

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

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

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

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

$R_p = 1 \cdot R_{eq,sdec,inv,p}$ 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 in which it is able to switch the pass transistor it has as a load. $C_d = C_{d,sdec,inv,p} + C_{d,sdec,inv,n}$ as usual is the self-load capacitance of the gate. $C_L = C_{g,rowpass}(N_{wl} + 2) + C_{g,slice}$ 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, plus one SST and one GST (hence the +2), plus the pass transistor which allows the read bit to go out from the block (hence the $+C_{g,slice}$).

2.1.3 | Row decoder

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 one. 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.

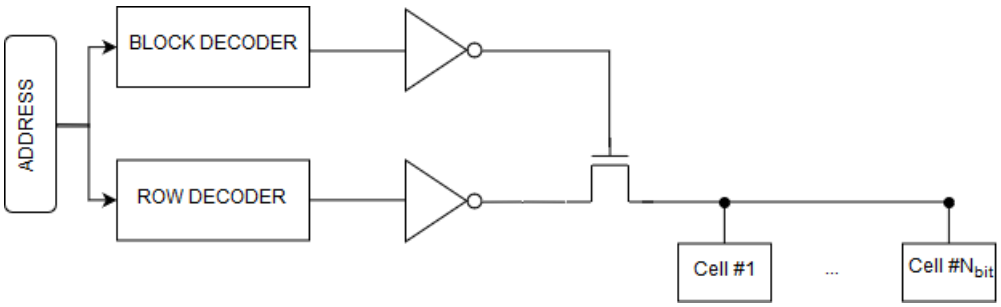

Figure 2.3: Row decoder and block decoder timing

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.

2.1.3.1 | 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, this time we have to use the Elmore model to represent the situation.

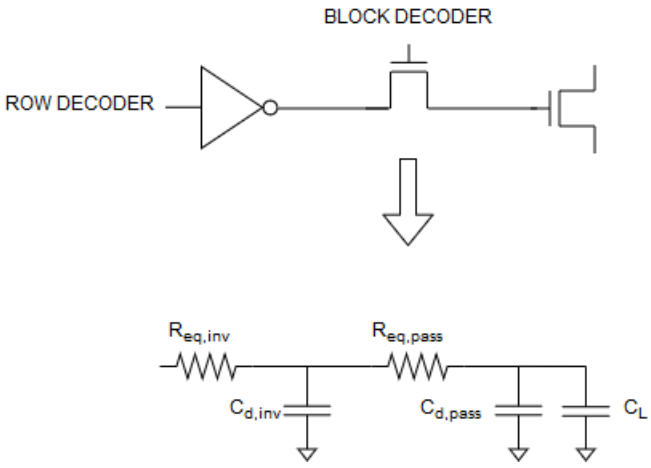

Figure 2.4: Elmore model for the wordline delay

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. Here we consider only the inverters driving the transistors GST (ground select transistor) and SST (string select transistor). Actually all the other lines coming out from the decoder are connected to different transistors, the floating gate transistors constituting the memory cells. Since the capacitance of a floating gate transistor is smaller than the one of a traditional transistor, the worst case is given for $C_L = C_{g,pt}N_{bl}$, where $C_{g,fg}$ is the gate capacitance of a GST or SST, and N_{bl} is the number of GST or SST per each wordline.

2.1.4 | Bit line delay

During the read operation, all the strings in the slice are connected to the bit line. Depending on the value memorized in the cell, they can discharge the bit line capacitance or act as an open circuit. The variation of the bitline capacitance provides the value of the cell which is measured through external components (sense amplifiers). The duration of the evaluation phase of the charge is the same for both 1s and 0s. It is timed accordingly with the delay in the discharge phase because, if the string act as an open circuit, the capacitance of the bit line remains the same, so there is no delay. The strings are read in parallel, hence the delay is the one corresponding to a single pillar.

An Elmore model has been used for the string where each FGT and SST/GST is modelled with a capacitance and a resistance as shown in the picture below.

So, in this phase, the delay has been evaluated by using the following formula:

$$\tau_{eval} = C_1 \cdot R_1 + C_2 \cdot (R_1 + R_2) + C_3 \cdot (R_1 + R_2 + R_3) + ...$$

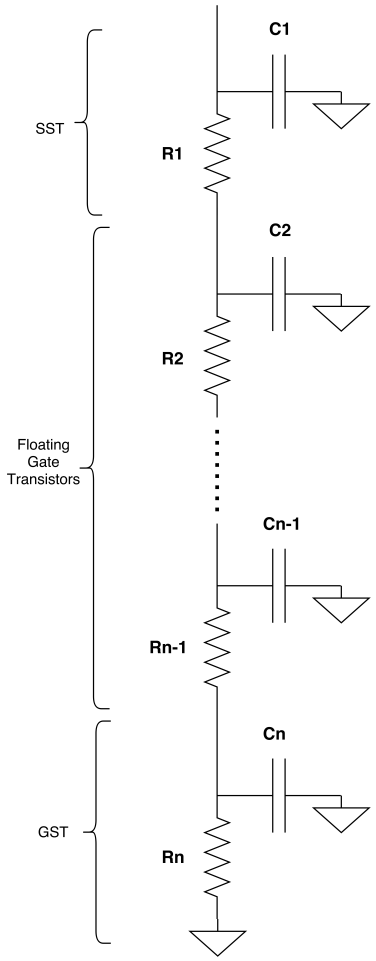

Figure 2.5: Elmore model for the bitline delay

We assume, however, that only a small part of this delay will be really needed in the estimation of the total delay, since we have a sense amplifier connected at the end of each bitline. The sense amplifier is at the beginning in a metastable state, but as soon as it senses a voltage difference between its input (the bitline) and the reference value provided from the external, it switches to a stable state, forcing at the same time a fast charge/discharge of the input line itself. So, the formula actually used in our computations includes also a generic K_{SA} factor (which may be 5%, for example):

$$\tau_{eval} = K_{SA}(C_1 \cdot R_1 + C_2 \cdot (R_1 + R_2) + C_3 \cdot (R_1 + R_2 + R_3) + ...)$$

2.1.5 | 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 bitline. Note that in this amplifier, input and output are somehow corresponding. The schematic of the component is shown below:

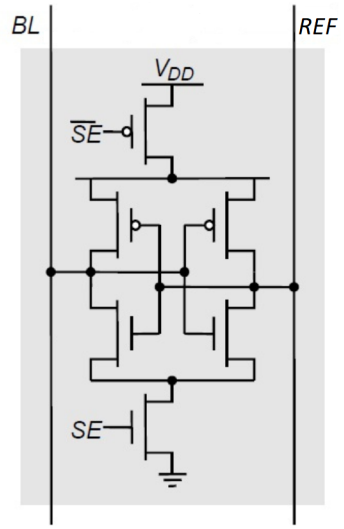

Figure 2.6: Sense amplifier structure

Its delay is described by a simple RC model:

$$\tau = R_{eq,sa,mod,parallel}(C_{d,sa,p} + C_{d,sa,n} + C_{g,sa,p} + C_{g,sa,n} + (C_{bl,wire} \cdot L_{bl}) + C_{d,colpass})$$

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 bitline and whose gate terminal is driven by the column decoder.

2.1.6 | Delay of the column pass transistor and of the slice transistor

Finally, the last contribution to the delay is given by the two pass transistors we have before the output from the block. The former is driven by the column decoder (by the inverter on its output), the latter by the block decoder (by the inverter on its output). We don't consider the delay of the column decoder, because it works together with the block decoder and the row decoder, and even if its delay was longer than the largest between the other two, we have also all the contributions from the charging of the wordline and from the discharging of the bitline and the switching of the sense amplifier, so at this point the output of the column decoder will have for sure become stable.

To compute the delay due to the two pass transistors we use again the Elmore model:

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

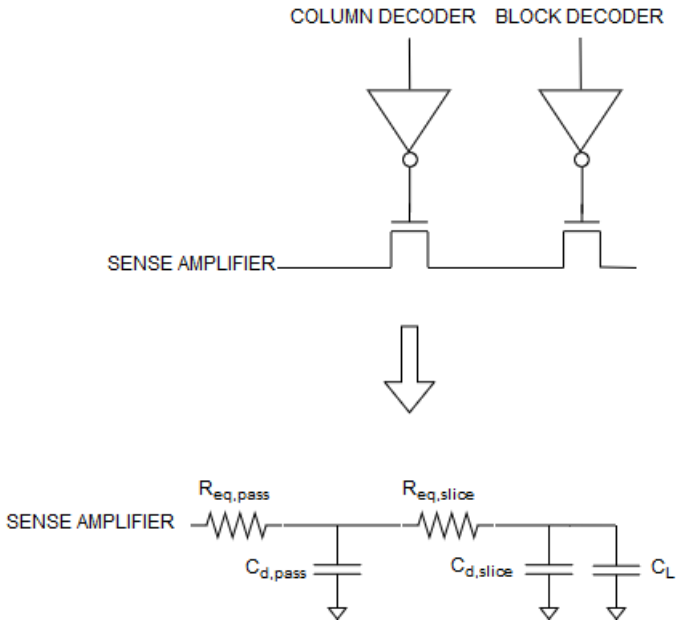

Figure 2.7: Elmore model for the pass transistors delay

$R_{eq,pass} = R_{eq,colpass}$ is the equivalent resistance of the pass transistor driven by the column decoder, whereas $C_{d,pass} = C_{d,colpass}$ is its self-load capacitance. $R_{eq,slice}$ and $C_{d,slice}$ are the analogous parameters for the pass transistor driven by the inverter out from the block decoder. C_L is unknown and in our analysis is assumed to be an open circuit.

2.1.7 | 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.

2.2 | Simulation result

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 wordlines and bitlines change in a well defined range, to analyse how the delay changes if we vary the dimensions of the memory array. In particular, the array of values we considered for both N_{wl} and N_{bl} is $[64, 128, 256, 512, 1024, 2048]$. For each simulation point we assumed that $N_{bl} = N_{wl}$, because usually the memory arrays are made as square as possible, for reasons of space availability on the board.

The result we obtained is reported in the figure below.

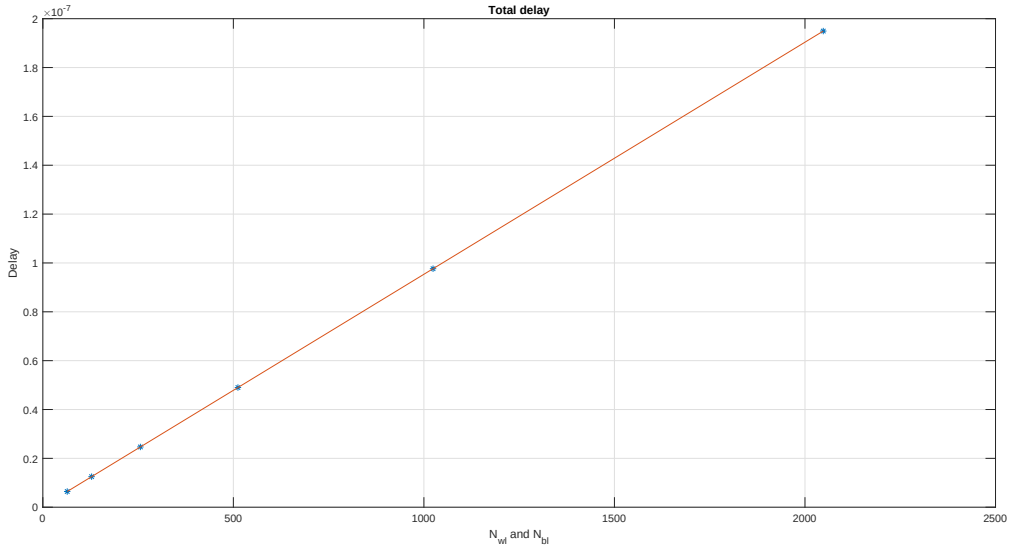

Figure 2.8: Simulation of the delay varying the size of the memory array

The behaviour represented is reasonable: in all the contributions previously discussed we always have at most a linear dependence on N_{wl} or N_{bl}; some terms are independent from the variations of these two parameters and they partially contribute to the offset that can be observed on the 64x64 simulation point. Instead, we never have a square dependence on either of the two parameters, so an overall linear behaviour is perfectly reasonable.

So, with the values chosen for the parameters involved, the total delay of a precharge&read operation spans between 6.43ns, in the 64x64 case, and 195ns in the 2048x2048 case.

Power Analysis

3

The estimation of power consumption is based on the analytical power model described in [1].

3.1 | Capacitance modeling

The considered capacitances are the following:

- $C_{g,fg}$ gate capacitance of floating gate transistor
- $C_{d,fg}$ "drain/source" capacitance of floating gate transistor
- $C_{g,pt}$ gate capacitance of a generic pass transistor (SST and GST)
- $C_{d,pt}$ "drain/source" capacitance of a generic pass transistor (SST and GST)
- $C_{g,rowpass}$, $C_{g,colpass}$ and $C_{g,slice}$gate capacitance of SET and column pass transistors
- $C_{d,rowpass}$, $C_{d,colpass}$ and $C_{d,slice}$ drain capacitance of SET, column and slice pass transistors
- $C_{wl,wire}$ wire capacitance per unit length of wordline
- $C_{bl,wire}$ wire capacitance per unit length of bitline
- $C_{ssl,wire}$ wire capacitance per unit length of string select line (the same for ground select line)
- $C_{slice,dec}, C_{row,dec}, C_{col,dec}$ total capacitances of each type of decoder
- $C_{SA,in}$ C_{SA} input and load capacitance of the sense amplifier
- $C_{g,pre}$ gate capacitance of the precharge transtistor
- C_{ext,pu_driver} external precharge unit driver capacitance

Other useful parameters are:

- N_{bl} number of bitlines
- N_{wl} number of wordlines
- L_{bl} length of bitline
- L_{wl} length of wordline
- N_{slice} number of slices
- N_{erase} number of erase cycles

3.1.1 | Floating gate transistor capacitances

To evaluate power dissipation of a Floating Gate Transistor, the values of its capacitances are needed.
While in a traditional MOS structure there are Source, Drain and Gate capacitances, in a FGT there is a double gate structure (floating gate and control gate): the two capacitances C_{fg} and C_{cg} can be modeled as a single equivalent gate capacitance $C_{g,fg}$, series of the two and then equal to:

$$C_{g,fg} = \frac{C_{fg} \cdot C_{cg}}{C_{fg} + C_{cg}}$$

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:

$$C_{wl} = C_{d,rowpass} + C_{g,fg} \cdot N_{bl} + C_{wl,wire} \cdot L_{wl}$$

$$C_{bl} = 2 \cdot C_{d,pt} + N_{wl} \cdot C_{d,fg} + C_{bl,wire} \cdot L_{bl} + C_{SA,in}$$

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:

$$C_{slice,dec} = C_{d,sdec,pcharge} + C_{d,sdec,eval} + C_{g,sdec,inv_p} + C_{g,sdec,inv_n}$$

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

where $C_{g,dec,eval}, C_{d,sdec,pcharge}$ and $C_{d,sdec,eval}$ 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, $C_{d,sdec,eval} = Block\_Address \cdot C_{d,sdec,n}$.
$C_{slice,stack}$ 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 and column decoders.

3.1.4 | Sense Amplifier

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

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

$$C_{SA} = C_{d,colpass} + C_{d,slice}$$

3.2 | Read Dynamic Power

3.2.1 | Decoding stage

During a read operation, the slice decoder selects the slice in which the operation has to be carried out. At the same time other two decoders, the row and the column one, are working to select the addressed bit. The formulas to calculate energy consumption in the decoding phase are the following:

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

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

$$E_{row,dec} = 0.5 \cdot C_{row,dec} \cdot (V_{rd,sel}^2 + V_{rd,unsel}^2 \cdot (N_{wl} - 1) + 2 \cdot V_{on,pt}^2) \cdot N_{slice}$$
$$E_{stack,rdec} = 0.5 \cdot C_{row,stack} \cdot V_{on,pt}^2 \cdot N_{slice}$$

$$E_{col,dec} = 0.5 \cdot C_{col,dec} \cdot V_{on,pt}^2 \cdot N_{slice}$$
$$E_{stack,cdec} = 0.5 \cdot C_{col,stack} \cdot V_{on,pt}^2 \cdot N_{slice}$$

In this analysis, we are making the assumption that all the decoders have the same structure, but in reality the page decoder has a more complex one, having to give different voltages in output for each wordline.
The energy consumption linked to the selection of the slice and of the column can be expressed as:

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

$$E_{col,pt} = 0.5 \cdot C_{g,colpass} \cdot V_{on,pt}^2$$

3.2.2 | Precharge

In this state all the lines are isolated from the memory array and the bitlines are biased to a specific value $V_{bl,prec}$, by the precharge block, to speed up the following operation in order to reduce the latency. The other lines are biased to ground.
The formulas to calculate dissipated energy in the precharge phase are the following:

$$E_{pre} = 0.5 \cdot (C_{ext,pu\_driver} + C_{g,pre}) \cdot V_{bl,prec}^2 \cdot N_b l$$

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

3.2.3 | Read operation

The wordline of the selected page is biased to ground ($V_{rd,sel}$) while the voltage on the insulated page is set to $V_{rd,unsel}$. In this way the unselected pages act as a transfer gates and are always on independently from the values stored in the FGTs. It is assumed that the initial voltage of the wordline is 0.

The energy to switch the selected wordline is the following:

$$E_{sel} = 0.5 \cdot C_{wl} \cdot V_{rd,sel}^2$$

The energy to switch the unselected wordlines is:

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

The bitlines are at the precharge voltage $V_{bl,prec}$ and are connected to the strings using SSTs and GSTs. Depending on the threshold voltage of the FGTs (1 or 0 stored) in the selected page, the bitlines can have two different kinds of voltage drop, $V_{rd,1}$ and $V_{rd,0}$. So a parameter that considers the distribution of 0s and 1s stored in the memory is used (p_0).

The energy to read a 1 is:

$$E_1 = 0.5 \cdot C_{bl} \cdot (V_{bl,prec} - V_{rd,1})^2 \cdot N_{bl} \cdot (1 - p_0)$$

The energy to read a 0 is:

$$E_0 = 0.5 \cdot C_{bl} \cdot (V_{bl,prec} - V_{rd,0})^2 \cdot N_{bl} \cdot p_0$$

where $V_{bl,prec} - V_{rd,0}$, $V_{bl,prec} - V_{rd,1}$ are the voltage swing to the read of '0' and '1'.
The energy to activate SST and GST is:

$$E_{pt} = 2 \cdot [0.5 \cdot C_{G,pt} \cdot V_{on,pt}^2] \cdot N_{bl}$$

where $V_{on,pt}$ is the voltage to enable the pass transistors.
The energy on the string select line and ground select line is:

$$E_{sl} = E_{ssl} + E_{gsl} = 2 \cdot [0.5 \cdot (C_{ssl,wire} \cdot L_{wl}) \cdot V_{on,pt}^2]$$

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:

$$E_{SA} = 0.5 \cdot C_{SA} \cdot V_{bl,prec} \cdot ((V_{bl,prec} - V_{rd,0}) \cdot p_0 + (V_{bl,prec} - V_{rd,1}) \cdot (1 - p_0)) \cdot N_b l$$

3.2.5 | Total Power

Total 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

In the write stage, only the cells where the logical 0s will be written must be programmed, the others need to be inhibited. This can be performed by applying a voltage on the bitline that is 80% of the voltage on the control gate ($V_{inhibit}$). The bitlines that need to be programmed with logical 0 are biased to ground. On all the control gates of the page a program voltage (V_{prog}) is applied. There is also a contribute due to tunneling energy E_{tunnel}. It is assumed that the precharge voltage of the wordline is 0.

For the writing operation the column decoding is unneeded, being the page the smallest programmable unit. So the energy consumption for the decoding stage is given, as before, by the relations:

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

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

$$E_{row,dec} = 0.5 \cdot C_{row,dec} \cdot (V_{prog}^2 + V_{inhibit}^2 \cdot (N_{wl} - 1) + 2 \cdot V_{on,pt}^2) \cdot N_{slice}$$
$$E_{stack,rdec} = 0.5 \cdot C_{row,stack} \cdot V_{on,pt}^2 \cdot N_{slice}$$

The energy to precharge the bitlines is:

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

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

The energy to switch the selected wordline is:

$$E_{sel} = 0.5 \cdot C_{wl} \cdot V_{prog}^2$$

The energy to switch the unselected wordlines is:

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

The inhibit energy to maintain 1 in the cells is:

$$E_{bl,inhibit} = 0.5 \cdot (C_{bl} - C_{d,fg} \cdot N_{wl}) \cdot (0.8 \cdot V_{inhibit})^2 \cdot N_{bl} \cdot (1 - p_0)$$

According to the self-boosted program inhibit model, the channel voltage is boosted to about 80% of the applied control gate voltage by biasing the bit-lines corresponding to logical 1 at a specific voltage, in order to have that the boosted channel voltage is a fraction of the applied control gate voltage ($0.8 \cdot V_{inhibit}$).

The energy to program (write a 0) is:

$$E_{bl,sel} = (0.5 \cdot (C_{bl} - C_{d,fg} \cdot N_{wl}) \cdot (0 - V_{bl,prec})^2 + E_{tunnel}) \cdot N_{bl} \cdot p_0$$

The program voltage to be applied to the bitline is 0.

The energy to activate the pass transistors is:

$$E_{pt} = 2 \cdot [0.5 \cdot C_{G,pt} \cdot V_{on,pt}^2] \cdot N_{bl}$$

The energy on the string select line and ground select line is:

$$E_{sl} = E_{ssl} + E_{gsl} = 2 \cdot [0.5 \cdot (C_{ssl,wire} \cdot L_{wl}) \cdot V_{on,pt}^2]$$

In conclusion, the total amount of energy for a write operation is given by the addition of all these terms.
Assuming f_{write} as write frequency and E_{write}as total write energy, total write power can be computed as follows:

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

3.4 | Erase power

For the erase operation, in planar flash NAND, the well of the block to erase is biased to an high voltage and the control gates are connected to ground. A high voltage is applied on the bitlines (V_{prog}) and the gates are biased to ground to have a tunnel effect that is in the opposite direction with respect to the write case. Also in this instance the tunnel energy must be considered.

Even in this case, the column decoding is unneeded and so the energy consumption is given, as before, by the relations:

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

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

$$E_{row,dec} = 0.5 \cdot C_{row,dec} \cdot (V_{bl,erase}^2 \cdot N_{wl} + 2 \cdot V_{on,pt}^2) \cdot N_{slice}$$
$$E_{stack,rdec} = 0.5 \cdot C_{row,stack} \cdot V_{on,pt}^2 \cdot N_{slice}$$

The precharge step is required:

$$E_{pre} = 0.5 \cdot C_{g,pre} \cdot V_{bl,prec}^2 \cdot N_b l$$

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

The energy necessary to start erasing the block is:

$$E_{erase,slice} = 0.5 \cdot C_{bl} \cdot (V_{bl,erase} - V_{bl,prec})^2 \cdot N_{bl} + E_{tunnel} \cdot N_{bl} \cdot N_{wl}$$

where $E_{tunnel} \cdot N_{bl} \cdot N_{wl}$ is the tunneling energy for the block.
The energy to activate the two pass transistors is the following:

$$E_{pt} = 2 \cdot [0.5 \cdot C_{G,pt} \cdot V_{on,pt}^2] \cdot N_{bl}$$

The energy on the string select line and ground select line is:

$$E_{sl} = E_{ssl} + E_{gsl} = 2 \cdot [0.5 \cdot (C_{ssl,wire} \cdot L_{wl}) \cdot V_{on,pt}^2]$$

Total energy for the erase operation is:

$$E_{erase} = E_{slice,dec} + E_{stack,sdec} + E_{row,pt} + E_{stack,rdec} + E_{row,dec} + E_{bl} + (E_{erase,slice} + E_{pt} + E_{sl}) \cdot N_{erase}$$

where N_{erase} is the number of erase cycles, necessary for the operation to be performed correctly.
Assuming f_{erase} as erase frequency, total erase power can be computed as follows:

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

3.5 | Simulation results

To verify the plausibility of our computations we assigned a reasonable value to each of the parameters involved in the equations.
In a first simulation the number of wordlines and bitlines change in a well defined range, to analyse how the power consumption changes if we vary the dimensions of the memory array. In particular, the array of values we considered for both N_{wl} and N_{bl}, as for the delay, is $[64, 128, 256, 512, 1024, 2048]$. For each simulation point we assumed that $N_{bl} = N_{wl}$, because usually the memory arrays are made as square as possible, for reasons of space availability on the board.

The obtained results are reported in the following figures.

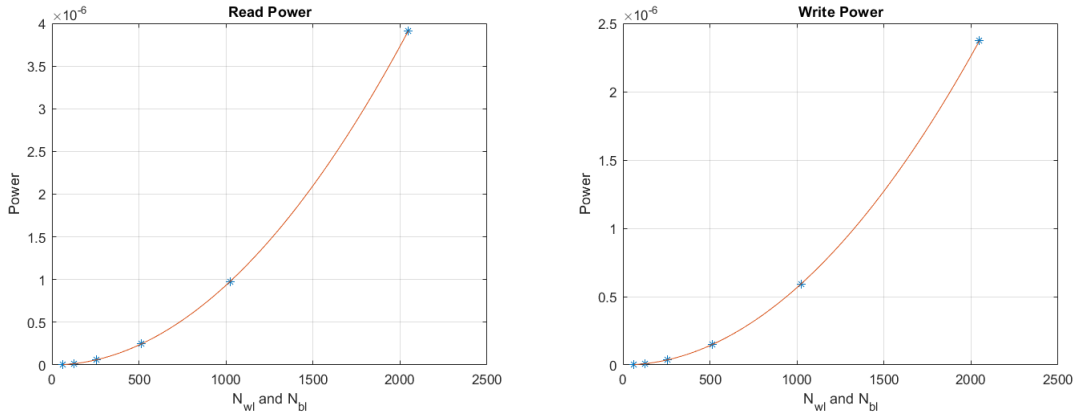

Figure 3.1: Read Dynamic Power and Write Dynamic Power versus N_{wl}, N_{bl}

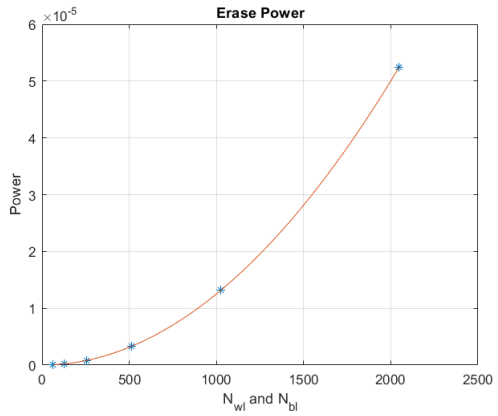

Figure 3.2: Erase Dynamic Power versus N_{wl}, N_{bl}

The behaviour represented is reasonable: in all the contributions previously discussed

there is a more than linear dependence on N_{wl} and N_{bl}. In particular the most relevant contributions are dependent from the product of these two parameters, giving as result a quadratic curve.
The curves in Figure 3.1 show that power consumption for read and write operations is in the order of some μW, while the erase power is about one order of magnitude larger.
In a second analysis N_{wl} and N_{bl} are fixed, with a value of 1024, while N_{slice} varies according to the values $[32, 64, 128, 256]$.

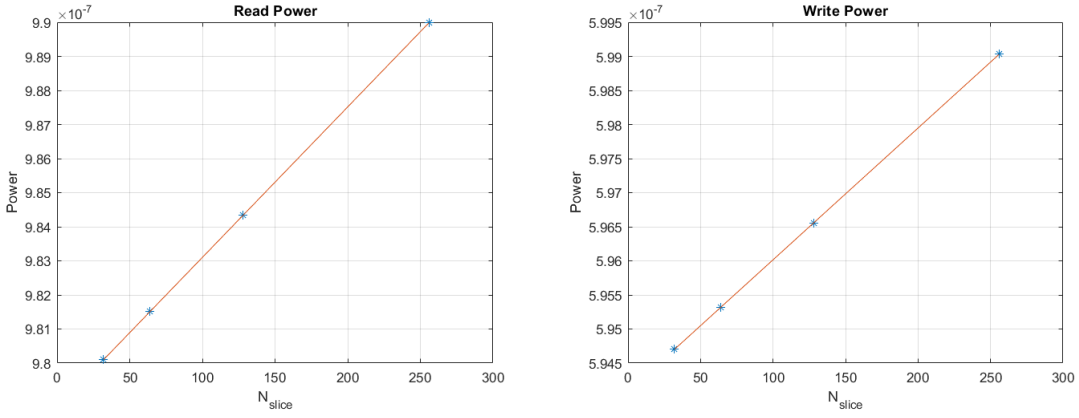

Figure 3.3: Read Dynamic Power and Write Dynamic Power versus N_{slice}

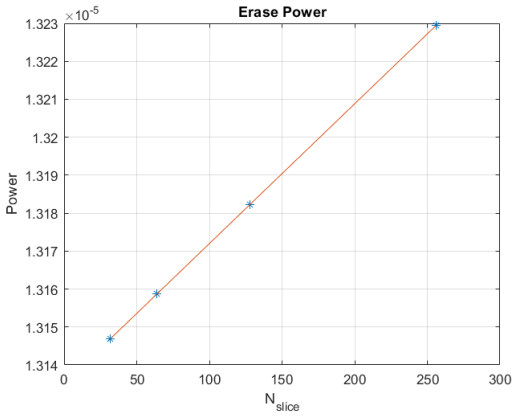

Figure 3.4: Erase Dynamic Power versus N_{slice}

As it can be seen, the dependency from N_{slice} is linear. However, having supposed that all the unselected blocks are deactivated, the increase in the curves is due to the power consumption of an higher number of decoders.

4 Area and Volume

In this chapter have been calculated area and volume of the 3D Memory complete structure (Memory, Decoders and Sence Amplifier).

4.1 | Memory array

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

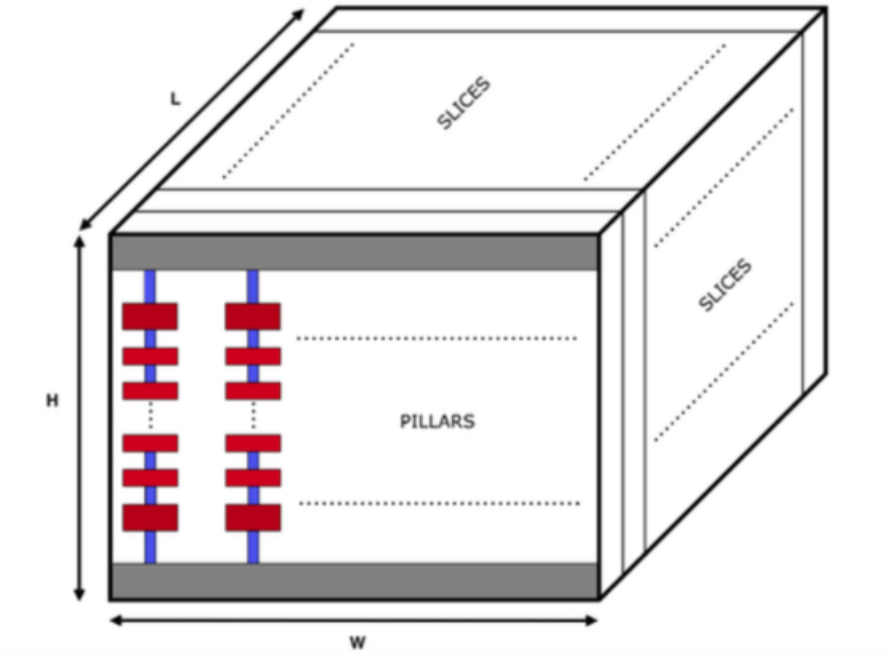

Figure 4.1: Simplified memory structure

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

- *Width* and *Length*: considering the symmetry of the model, the same pitch between two pillars in the same slice and between two pillars in two different slices has been used.

$$W = N_{column} \cdot Pitch_{pp} = N_{bl} \cdot Pitch_{pp}$$

$$L = N_{slice} \cdot Pitch_{pp}$$

- *Height of the stack*: the terms considered are the height of the MOS FGTs and MOS GST/SST, the pitch between contacts and GST/SST and between FGTs and GST/SST, the height of the S/D contacts, as shown in 4.2.

$$H = 2 \cdot Pitch_{contact-PT} + (N_{wl} - 1) \cdot Pitch_{FGT-FGT} + 2 \cdot Pitch_{PT-FGT} + 2 \cdot h_{contact}$$

Obviously, for planar MOS technology, $h_{contact}$ is set to zero.
Finally, the area and the volume of the array of memory can be calculated as:

$$\textbf{Memory\_\_Array\_\_Area} = H \cdot W$$

$$\textbf{Memory\_\_Array\_\_Volume} = H \cdot W \cdot L$$

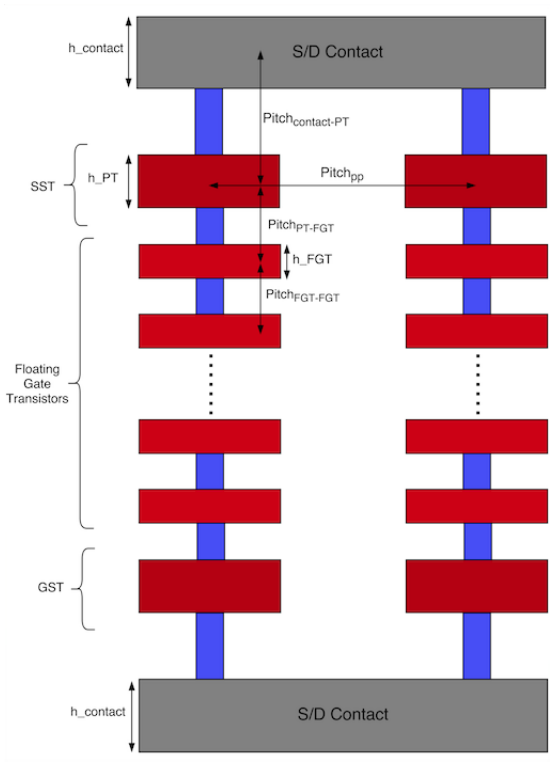

Figure 4.2: Stack structure

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 length. So:

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

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

$$\begin{aligned} \textbf{Block\_Dec\_Area} &= \#Tr\_n\_Block\_Dec \cdot Tr\_n\_Area + \\ &+ \#Tr\_p\_Block\_Dec \cdot Tr\_p\_Area \\ \textbf{Block\_Dec\_Volume} &= Block\_Dec\_Area \cdot L \end{aligned}$$

- **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 $(N_{wl} + 2)$. The volume is the area multiplied by the length. So:

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

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

$$\begin{aligned} \textbf{Row\_Dec\_Area} &= \#Tr\_n\_Row\_Dec \cdot Tr\_n\_Area + \\ &+ \#Tr\_p\_Row\_Dec \cdot Tr\_p\_Area \\ \textbf{Row\_Dec\_Volume} &= Row\_Dec\_Area \cdot L \end{aligned}$$

- **Column Decoder**: The area of the *Column Decoder* has been calculated as in *Row Decoder* but in this case the number of input is done by the *Column Address* while the output is equal to the number of bit line that is N_{bl}. The volume is the area multiplied by the length. So:

$$\#Tr\_n\_Column\_Dec = Column\_Address \cdot N_{bl} + N_{bl} + Column\_Address$$

$$\#Tr\_p\_Column\_Dec = 2 \cdot N_{bl} + Column\_Address$$

$$\textbf{Column\_Dec\_Area} = \#Tr\_n\_Column\_Dec \cdot Tr\_n\_Area +$$
$$+\#Tr\_p\_Column\_Dec \cdot Tr\_p\_Area$$
$$\textbf{Column\_Dec\_Volume} = Column\_Dec\_Area \cdot L$$

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

$$Block\_Address = ceil(log2(N_{slice}))$$
$$Row\_Address = ceil(log2(N_{wl}))$$
$$Column\_Address = ceil(log2(N_{bl}))$$

4.3 | 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.

- **Row Pass Transistors**: The area has been calculated as $(N_{wl} + 2)$ multiplied by the area of single transistor, while the volume is the area multiplied by the length. So:

$$\textbf{Pass\_Row\_Area} = (N_{wl} + 2) \cdot Tr\_n\_Area$$
$$\textbf{Pass\_Row\_Volume} = Pass\_Row\_Area \cdot L$$

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

$$\textbf{Pass\_Column\_Area} = N_{bl} \cdot Tr\_n\_Area$$

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

- **Slice Pass Transistors**: The area is the area of single transistor for each block, while the volume is the area multiplied by the length. So:

$$\textbf{Pass\_Slice\_Area} = N_{Slice} \cdot Tr\_n\_Area$$

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

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

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

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

4.4 | Sense Amplifier

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

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

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

$$\textbf{SA\_Area} = \#Tr\_n\_SA \cdot Tr\_n\_Area + \#Tr\_p\_SA \cdot Tr\_p\_Area$$

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

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:

$$\textbf{Total\_Area} = Memory\_Array\_Area + Block\_Dec\_Area + Row\_Dec\_Area+$$

$$+Column\_Dec\_Area + Pass\_Row\_Area + Pass\_Column\_Area+$$

$$Pass\_Slice\_Area + SA\_Area$$

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

$$\textbf{Total\_Volume} = Memory\_Array\_Volume + Block\_Dec\_Volume + Row\_Dec\_Volume+$$

$$+Column\_Dec\_Volume + Pass\_Row\_Volume + Pass\_Column\_Volume+$$

$$Pass\_Slice\_Volume + SA\_Volume$$

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 four graph in which are represented how the area and the volume change with the value of N_{wl} and N_{bl} and with the value of N_{slice}. In particular, the array of values it has been considered for both N_{wl} and N_{bl} is $[64, 128, 256, 512, 1024, 2048]$ while for N_{slice} is $[32, 64, 128, 256]$. For each simulation point, it has been assumed that $N_{bl} = N_{wl}$, because usually the memory arrays are made as square as possible, for reasons of space availability on the board.
The behaviours represented are reasonable: in the first two graphs 4.3 and 4.4 there is a square dependency N_{wl} and N_{bl}. Changing the value of these parameters, the total area and volume increase of one order of magnitude in the bigger case (2048x2048) respect to the the smaller one (64x64), with a maximum values of $2 \cdot 10^{-7} m^2$ for the Total Area and $6 \cdot 10^{-14} m^3$ for the Total Volume. In the other two graphs 4.5 and 4.6 there is a linear dependency N_{slice}. Changing the value of this parameter, the total area not increase significantly, while the volume increase of one order of magnitude. The maximum values are $5.21 \cdot 10^{-8} m^2$ for the Total Area and $4 \cdot 10^{-12} m^3$ for the Total Volume.
The results obtained are reported in the figures below.

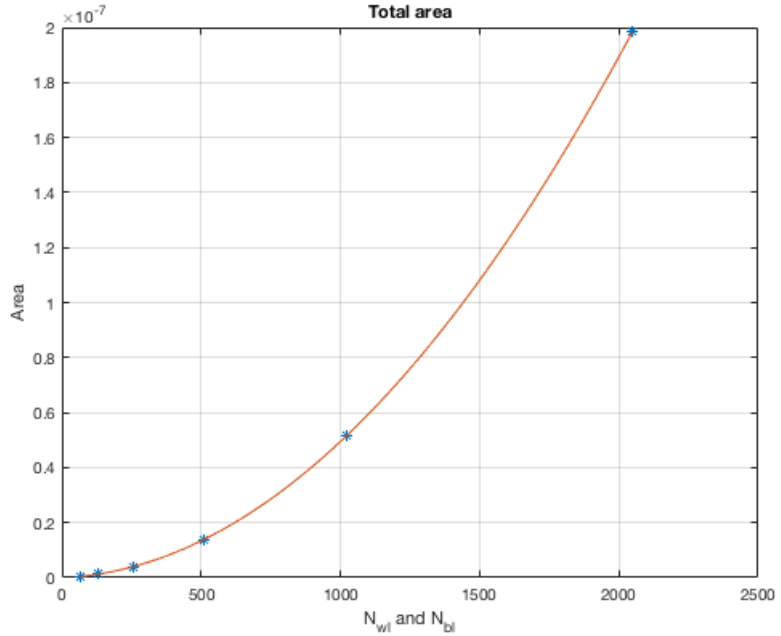

Figure 4.3: Simulation of the Total Area value, varying the size of the memory array

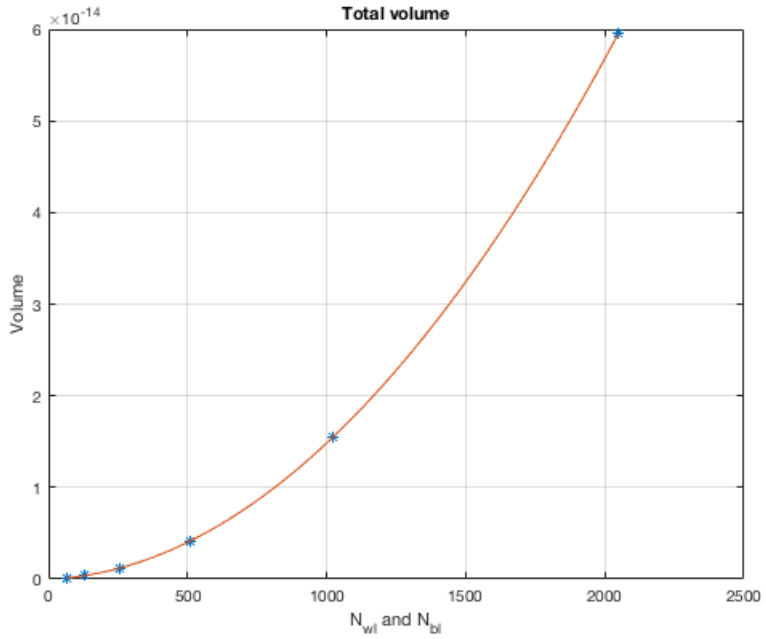

Figure 4.4: Simulation of the Total Volume value, varying the size of the memory array

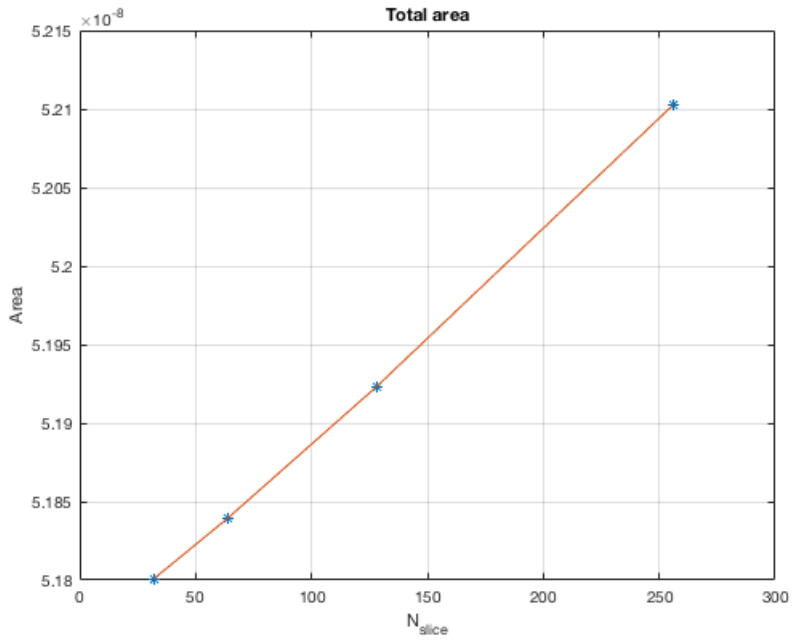

Figure 4.5: Simulation of the Total Area value, varying the number of slice

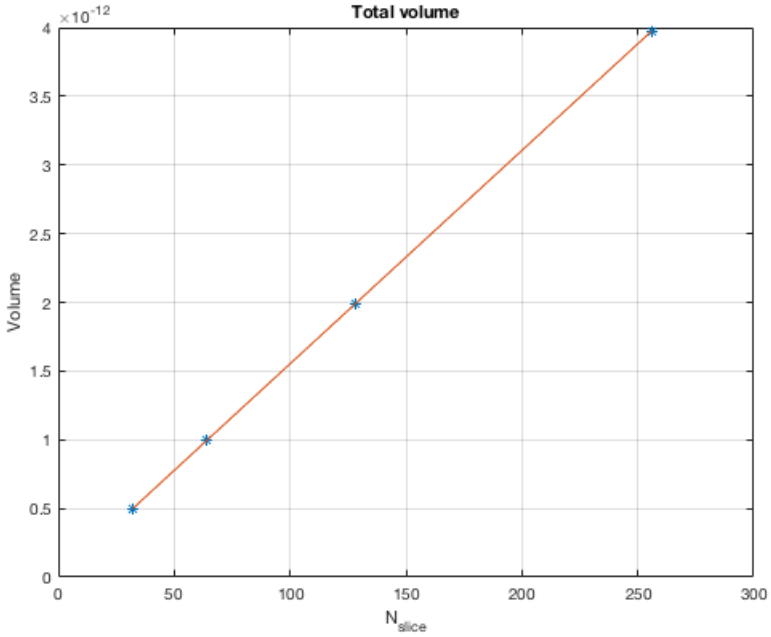

Figure 4.6: Simulation of the Total Volume value, varying the number of slice

5 Matlab code

5.1 | InputParameters.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
%transistors area
Tr_n_Area = W*L;
Tr_p_Area = Beta*Tr_n_Area;
%gate capacitance of a floating gate transistor
        C_g_fg = 2.5E-16*L;
%drain capacitance of a floating gate transistor
        C_d_fg = 1.7E-16*L;
%gate capacitance of the SST and GST transistors
        C_g_pt = 5E-16*L;
%drain capacitance of the SST and GST transistors
        C_d_pt = 1.7E-16*L;
%interconnect capacitance per unit of lenght of the wordline
        C_wl_wire = 3.06E-14;
%interconnect capacitance per unit of lenght of the bitline
        C_bl_wire = 3.06E-14;
%interconnect capacitance per unit of lenght of the string select line
        C_ssl_wire = 3.06E-14;
```

```
%number of bitlines
        N_bl_array = [64, 128, 256, 512, 1024, 2048];
%number of wordlines
        N_wl_array = [64, 128, 256, 512, 1024, 2048];
%number of slices
    N_slice_array = [32, 64, 128, 256];
%pass-transistor voltage
        V_on_pt = 3;
%precharge voltage
        V_bl_prec = 3;
%erase voltage
        V_bl_erase = 20;
%zero reading probability
        p_0 = 0.5;
%tunnel energy [J]
        E_tunnel = 1.6E-19;
%programming bitline voltage
        V_prog = 20;
%selected wordline voltage
        V_rd_sel = 3;
%unselected wordline voltage
        V_rd_unsel = 6;
%one read voltage
        V_rd_1 = 3;
%zero read voltage
        V_rd_0 = 2.5;
%voltage to inhibit the programming
    V_inhibit = 10;


%read frequency
        f_read = 5.00E+06;
%write frequency
        f_write = 1.00E+06;
%erase frequency
        f_erase = 5.00E+06;


%R e C are defined just before the call to the function


%ion current of the precharge mos
        I_on_driver = 1E-3*L;
```

```
73  %output capacitance of the driver driving the gate of the pmos inside the precharge
    ↪ unit
74          C_ext_pu_driver = 20E-16*L;
75  %gate capacitante of the pmos inside the precharge unit
76          Cg_pre = Beta*5E-16*L;
77  %output resistance of the driver driving the gate of the pmos inside the precharge
    ↪ unit
78          R_ext_pu_driver = 50;
79  %gate capacitance of a nmos inside the slice decoder
80          Cg_sdec_n = 5E-16*L;
81  %gate capacitance of a nmos inside the row decoder
82          Cg_rdec_n = 5E-16*L;
83  %gate capacitance of a nmos inside the column decoder
84          Cg_cdec_n = 5E-16*L;
85  %drain capacitance of the evaluation n-mos transistor in the slice decoder
86          Cd_sdec_n = 1.7E-16*L;
87  %drain capacitance of the evaluation n-mos transistor in the row decoder
88          Cd_rdec_n = 1.7E-16*L;
89  %drain capacitance of the evaluation n-mos transistor in the column decoder
90          Cd_cdec_n = 1.7E-16*L;
91  %drain capacitance of the precharge pmos inside the dynamic slice decoder
92          Cd_sdec_pcharge = Beta*1.7E-16*L;
93  %gate capacitance of the pmos inside the inverter on the output of the slice decoder
94          Cg_sdec_inv_p = Beta*5E-16*L;
95  %gate capacitance of the nmos inside the inverter on the output of the slice decoder
96          Cg_sdec_inv_n = 5E-16*L;
97  %equivalent resistance of the nmos in the slice decoder
98          Req_sdec_n = 200;
99  %equivalent resistance of the pmos inside the inverter on the output of the slice
    ↪ decoder
100         Req_sdec_inv_p = 200;
101 %drain resistance of the pmos inside the inverter on the output of the slice decoder
102         Cd_sdec_inv_p = Beta*1.7E-16*L;
103 %drain resistance of the pmos inside the inverter on the output of the slice decoder
104         Cd_sdec_inv_n = 1.7E-16*L;
105 %equivalent resistance of the nmos in the row decoder
106         Req_rdec_n = 200;
107 %drain capacitance of the precharge pmos inside the dynamic row decoder
108         Cd_rdec_pcharge = Beta*1.7E-16*L;
109 %gate capacitance of the pmos inside the inverter on the output of the row decoder
110         Cg_rdec_inv_p = Beta*5E-16*L;
111 %gate capacitance of the nmos inside the inverter on the output of the row decoder
112         Cg_rdec_inv_n = 5E-16*L;
```

```
113 %equivalent resistance of the pmos inside the inverter on the output of the row
    ↪ decoder
114         Req_rdec_inv_p = 200;
115 %drain capacitance of the p_MOS of the inverter on the output of the row decoder
116         Cd_rdec_inv_p = Beta*1.7E-16*L;
117 %drain capacitance of the n_MOS of the inverter on the output of the row decoder
118         Cd_rdec_inv_n = 1.7E-16*L;
119 %drain capacitance of the precharge MOS inside the dynamic column decoder
120         Cd_cdec_pcharge = Beta*1.7E-16*L;
121 %gate capacitance of the p_MOS of the inverter on the output of the row decoder
122         Cg_cdec_inv_p = Beta*5E-16*L;
123 %gate capacitance of the n_MOS of the inverter on the output of the row decoder
124         Cg_cdec_inv_n = 5E-16*L;
125 %drain capacitance of the sense amplifier p-MOS transistor
126         Cd_sa_p = Beta*1.7E-16*L;
127 %drain capacitance of the sense amplifier n-MOS transistor
128         Cd_sa_n = 1.7E-16*L;
129 %gate capacitance of the sense amplifier p-MOS transistor
130         Cg_sa_p = Beta*5E-16*L;
131 %gate capacitance of the sense amplifier n-MOS transistor
132         Cg_sa_n = 5E-16*L;
133 %input sense amplifier resistance
134         Req_sa_mod_parallel = 100;
135 %percentage of voltage swing that makes the sense amplifier switch
136         K_SA = 0.05;
137 %equivalent resistance of the slice pass transistor
138         Req_slice = 200;
139 %drain capacitance of the pass transistor on the output from the slice
140         Cd_slice = 1.7E-16*L;
141 %gate capacitance of the pass transistor on the output from the slice
142         Cg_slice = 5E-16*L;
143 %equivalent resistance of a row pass transistor, on the output of the row decoder and
    ↪ connecting it to the wordline
144         Req_rowpass = 200;
145 %gate capacitance of a row pass transistor, on the output of the row decoder and
    ↪ connecting it to the wordline
146         Cg_rowpass = 5E-16*L;
147 %drain capacitance of a row pass transistor, on the output of the row decoder and
    ↪ connecting it to the wordline
148         Cd_rowpass = 1.7E-16*L;
149 %equivalent resistance of a column pass transistor, on the output of the column
    ↪ decoder and connecting it to the sense amplifier
150         Req_colpass = 200;
```

```
%gate capacitance of the pass transistor on the output of the column decoder and connecting it to the sense amplifier
        Cg_colpass = 5E-16*L;
%drain capacitance of the pass transistor on the output of the column decoder and connecting it to the sense amplifier
        Cd_colpass = 1.7E-16*L;
%number of erase cycles
        N_erase = 3;

%distance between columns of transistor
        Pitch_pp = 3.00E-07;
%gate/drain distance
        Pitch_contact_PT = 5.00E-08;
%Gate gate distance between floating gate mos
        Pitch_FGT_FGT = 1.50E-07;
%Gate gate distance between floating gate mos and normal mos
        Pitch_PT_FGT = 1.50E-07;
%drain/source contact in new generation mos (equal to 0 for bulk tech)
        H_contact = 0;


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

%%COMPUTATIONS - TEST 1 N_bl N_wl Variation

for i=1:numel(N_wl_array)
N_slice = 1;
N_wl = N_wl_array(i);
N_bl = N_bl_array(i);

for j=1:N_wl
%array of the resistances used in the elmore delay model of a string
        R(j)= 200;
%array of the capacitances used in the elmore delay model of a string
        C(j)= C_d_fg;
end


 [ P_read, P_write, P_erase, Total_area, Total_volume, Total_delay ] = Memories3D (        N_slice,...
        C_g_fg,...
        C_d_fg,...
```

```
191         C_g_pt,...
192         C_d_pt,...
193         C_wl_wire,...
194         C_bl_wire,...
195         C_ssl_wire,...
196         N_bl,...
197         N_wl,...
198         V_on_pt,...
199         V_bl_prec,...
200     V_bl_erase,...
201         p_0,...
202         E_tunnel,...
203         V_prog,...
204         V_rd_sel,...
205         V_rd_unsel,...
206         V_rd_1,...
207         V_rd_0,...
208         V_inhibit,...
209         f_read,...
210         f_write,...
211         f_erase,...
212         R,...
213         C,...
214         I_on_driver,...
215         C_ext_pu_driver,...
216         Cg_pre,...
217         R_ext_pu_driver,...
218         Cg_sdec_n,...
219         Cg_rdec_n,...
220         Cg_cdec_n,...
221         Cd_sdec_n,...
222         Cd_rdec_n,...
223         Cd_cdec_n,...
224         Req_sdec_n,...
225         Cd_sdec_pcharge,...
226         Cg_sdec_inv_p,...
227         Cg_sdec_inv_n,...
228         Req_sdec_inv_p,...
229         Cd_sdec_inv_p,...
230         Cd_sdec_inv_n,...
231         Req_rdec_n,...
232         Cd_rdec_pcharge,...
233         Cg_rdec_inv_p,...
```

```
234         Cg_rdec_inv_n,...
235         Req_rdec_inv_p,...
236         Cd_rdec_inv_p,...
237         Cd_rdec_inv_n,...
238         Cd_cdec_pcharge,...
239         Cg_cdec_inv_p,...
240         Cg_cdec_inv_n,...
241         Cd_sa_p,...
242         Cd_sa_n,...
243         Cg_sa_p,...
244         Cg_sa_n,...
245         Req_sa_mod_parallel,...
246         K_SA,...
247         Req_slice,...
248         Cd_slice,...
249         Cg_slice,...
250         Req_rowpass,...
251         Cg_rowpass,...
252         Cd_rowpass,...
253         Req_colpass,...
254         Cg_colpass,...
255         Cd_colpass,...
256         N_erase,...
257         Tr_n_Area,...
258         Tr_p_Area,...
259         Pitch_pp,...
260         Pitch_contact_PT,...
261         Pitch_FGT_FGT,...
262         Pitch_PT_FGT,...
263         H_contact);
264
265 Total_delay_array_1(i)=Total_delay;
266 Total_area_array_1(i)=Total_area;
267 Total_volume_array_1(i)=Total_volume;
268 P_read_array_1(i)=P_read;
269 P_write_array_1(i)=P_write;
270 P_erase_array_1(i)=P_erase;
271
272 end
273
274 figure(1)
275 xq= 64:1:2048;
276 s = spline(N_wl_array,Total_delay_array_1,xq); %xq coordinate x punti interpolazione,
    ↪  s coordinate y punti interpolazione
```

```
277 plot(N_wl_array,Total_delay_array_1, '*', xq, s)
278 title('Total delay')
279 xlabel('N_{wl} and N_{bl}')
280 ylabel('Delay')
281 grid on
282
283 figure(2)
284 xq= 64:1:2048;
285 s = spline(N_wl_array,Total_area_array_1,xq); %xq coordinate x punti interpolazione,
    ↪ s coordinate y punti interpolazione
286 plot(N_wl_array,Total_area_array_1, '*', xq, s)
287 title('Total area')
288 xlabel('N_{wl} and N_{bl}')
289 ylabel('Area')
290 grid on
291
292 figure(3)
293 xq= 64:1:2048;
294 s = spline(N_wl_array,Total_volume_array_1,xq); %xq coordinate x punti
    ↪ interpolazione, s coordinate y punti interpolazione
295 plot(N_wl_array,Total_volume_array_1, '*', xq, s)
296 title('Total volume')
297 xlabel('N_{wl} and N_{bl}')
298 ylabel('Volume')
299 grid on
300
301 figure(4)
302 xq= 64:1:2048;
303 s = spline(N_wl_array,P_read_array_1,xq); %xq coordinate x punti interpolazione, s
    ↪ coordinate y punti interpolazione
304 plot(N_wl_array,P_read_array_1, '*', xq, s)
305 title('Read Power')
306 xlabel('N_{wl} and N_{bl}')
307 ylabel('Power')
308 grid on
309
310 figure(5)
311 xq= 64:1:2048;
312 s = spline(N_wl_array,P_write_array_1,xq); %xq coordinate x punti interpolazione, s
    ↪ coordinate y punti interpolazione
313 plot(N_wl_array,P_write_array_1, '*', xq, s)
314 title('Write Power')
315 xlabel('N_{wl} and N_{bl}')
```

```
ylabel('Power')
grid on

figure(6)
xq= 64:1:2048;
s = spline(N_wl_array,P_erase_array_1,xq); %xq coordinate x punti interpolazione, s coordinate y punti interpolazione
plot(N_wl_array,P_erase_array_1, '*', xq, s)
title('Erase Power')
xlabel('N_{wl} and N_{bl}')
ylabel('Power')
grid on

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

%%COMPUTATIONS - TEST 2 N_slice Variation

for k=1:numel(N_slice_array)
N_slice = N_slice_array(k);
N_wl = N_wl_array(5);
N_bl = N_bl_array(5);

for l=1:N_wl
%array of the resistances used in the elmore delay model of a string
        R(l)= 200;
%array of the capacitances used in the elmore delay model of a string
        C(l)= C_d_fg;
end


 [ P_read, P_write, P_erase, Total_area, Total_volume, Total_delay ] = Memories3D (        N_slice,...
        C_g_fg,...
        C_d_fg,...
        C_g_pt,...
        C_d_pt,...                %c_gaa_pt PASS TRANSISTOR SST GST
        C_wl_wire,...
        C_bl_wire,...
        C_ssl_wire,...
        N_bl,...
        N_wl,...
        V_on_pt,...
```

```
357         V_bl_prec,...
358     V_bl_erase,...
359         p_0,...
360         E_tunnel,...
361         V_prog,...
362         V_rd_sel,...
363         V_rd_unsel,...
364         V_rd_1,...
365         V_rd_0,...
366         V_inhibit,...
367         f_read,...
368         f_write,...
369         f_erase,...
370         R,...
371         C,...
372         I_on_driver,...
373         C_ext_pu_driver,...
374         Cg_pre,...
375         R_ext_pu_driver,...
376         Cg_sdec_n,...
377         Cg_rdec_n,...
378         Cg_cdec_n,...
379         Cd_sdec_n,...
380         Cd_rdec_n,...
381         Cd_cdec_n,...
382         Req_sdec_n,...
383         Cd_sdec_pcharge,...
384         Cg_sdec_inv_p,...
385         Cg_sdec_inv_n,...
386         Req_sdec_inv_p,...
387         Cd_sdec_inv_p,...
388         Cd_sdec_inv_n,...
389         Req_rdec_n,...
390         Cd_rdec_pcharge,...
391         Cg_rdec_inv_p,...
392         Cg_rdec_inv_n,...
393         Req_rdec_inv_p,...
394         Cd_rdec_inv_p,...
395         Cd_rdec_inv_n,...
396         Cd_cdec_pcharge,...
397         Cg_cdec_inv_p,...
398         Cg_cdec_inv_n,...
399         Cd_sa_p,...
```

```
400         Cd_sa_n,...
401         Cg_sa_p,...
402         Cg_sa_n,...
403         Req_sa_mod_parallel,...
404         K_SA,...
405         Req_slice,...
406         Cd_slice,...
407         Cg_slice,...
408         Req_rowpass,...
409         Cg_rowpass,...
410         Cd_rowpass,...
411         Req_colpass,...
412         Cg_colpass,...
413         Cd_colpass,...
414         N_erase,...
415         Tr_n_Area,...
416         Tr_p_Area,...
417         Pitch_pp,...
418         Pitch_contact_PT,...
419         Pitch_FGT_FGT,...
420         Pitch_PT_FGT,...
421         H_contact);
422
423 Total_delay_array_2(k)=Total_delay;
424 Total_area_array_2(k)=Total_area;
425 Total_volume_array_2(k)=Total_volume;
426 P_read_array_2(k)=P_read;
427 P_write_array_2(k)=P_write;
428 P_erase_array_2(k)=P_erase;
429
430 end
431
432 figure(7)
433 xq= 32:1:256;
434 s = interp1(N_slice_array,Total_delay_array_2,xq); %xq coordinate x punti
    ↪  interpolazione, s coordinate y punti interpolazione
435 plot(N_slice_array,Total_delay_array_2, '*', xq, s)
436 title('Total delay')
437 xlabel('N_{slice}')
438 ylabel('Delay')
439 grid on
440
441 figure(8)
```

```
442 xq= 32:1:256;
443 s = interp1(N_slice_array,Total_area_array_2,xq); %xq coordinate x punti
    ↪ interpolazione, s coordinate y punti interpolazione
444 plot(N_slice_array,Total_area_array_2, '*', xq, s)
445 title('Total area')
446 xlabel('N_{slice}')
447 ylabel('Area')
448 grid on
449
450 figure(9)
451 xq= 32:1:256;
452 s = interp1(N_slice_array,Total_volume_array_2,xq); %xq coordinate x punti
    ↪ interpolazione, s coordinate y punti interpolazione
453 plot(N_slice_array,Total_volume_array_2, '*', xq, s)
454 title('Total volume')
455 xlabel('N_{slice}')
456 ylabel('Volume')
457 grid on
458
459 figure(10)
460 xq= 32:1:256;
461 s = interp1(N_slice_array,P_read_array_2,xq); %xq coordinate x punti interpolazione,
    ↪ s coordinate y punti interpolazione
462 plot(N_slice_array,P_read_array_2, '*', xq, s)
463 title('Read Power')
464 xlabel('N_{slice}')
465 ylabel('Power')
466 grid on
467
468 figure(11)
469 xq= 32:1:256;
470 s = interp1(N_slice_array,P_write_array_2,xq); %xq coordinate x punti interpolazione,
    ↪ s coordinate y punti interpolazione
471 plot(N_slice_array,P_write_array_2, '*', xq, s)
472 title('Write Power')
473 xlabel('N_{slice}')
474 ylabel('Power')
475 grid on
476
477 figure(12)
478 xq= 32:1:256;
479 s = interp1(N_slice_array,P_erase_array_2,xq); %xq coordinate x punti interpolazione,
    ↪ s coordinate y punti interpolazione
```

```
480 plot(N_slice_array,P_erase_array_2, '*', xq, s)
481 title('Erase Power')
482 xlabel('N_{slice}')
483 ylabel('Power')
484 grid on
```

5.2 | Memories3D.m

```
1  function [ P_read, P_write, P_erase, Total_area, Total_volume, Total_delay ] =
   ↪  Memories3D (        N_slice,...
2          C_g_fg,...
3          C_d_fg,...
4          C_g_pt,...
5          C_d_pt,...
6          C_wl_wire,...
7          C_bl_wire,...
8          C_ssl_wire,...
9          N_bl,...
10         N_wl,...
11         V_on_pt,...
12         V_bl_prec,...
13     V_bl_erase,...
14         p_0,...
15         E_tunnel,...
16         V_prog,...
17         V_rd_sel,...
18         V_rd_unsel,...
19         V_rd_1,...
20         V_rd_0,...
21         V_inhibit,...
22         f_read,...
23         f_write,...
24         f_erase,...
25         R,...
26         C,...
27         I_on_driver,...
28         C_ext_pu_driver,...
29         Cg_pre,...
30         R_ext_pu_driver,...
```

```
31          Cg_sdec_n,...
32          Cg_rdec_n,...
33          Cg_cdec_n,...
34          Cd_sdec_n,...
35          Cd_rdec_n,...
36          Cd_cdec_n,...
37          Req_sdec_n,...
38          Cd_sdec_pcharge,...
39          Cg_sdec_inv_p,...
40          Cg_sdec_inv_n,...
41          Req_sdec_inv_p,...
42          Cd_sdec_inv_p,...
43          Cd_sdec_inv_n,...
44          Req_rdec_n,...
45          Cd_rdec_pcharge,...
46          Cg_rdec_inv_p,...
47          Cg_rdec_inv_n,...
48          Req_rdec_inv_p,...
49          Cd_rdec_inv_p,...
50          Cd_rdec_inv_n,...
51          Cd_cdec_pcharge,...
52          Cg_cdec_inv_p,...
53          Cg_cdec_inv_n,...
54          Cd_sa_p,...
55          Cd_sa_n,...
56          Cg_sa_p,...
57          Cg_sa_n,...
58          Req_sa_mod_parallel,...
59          K_SA,...
60          Req_slice,...
61          Cd_slice,...
62          Cg_slice,...
63          Req_rowpass,...
64          Cg_rowpass,...
65          Cd_rowpass,...
66          Req_colpass,...
67          Cg_colpass,...
68          Cd_colpass,...
69          N_erase,...
70          Tr_n_Area,...
71          Tr_p_Area,...
72          Pitch_pp,...
73          Pitch_contact_PT,...
```

```
        Pitch_FGT_FGT,...
        Pitch_PT_FGT,...
        H_contact)



%% MEMORY PARAMETERS - COMPUTED %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% GEOMETRY PARAMETERS - COMPUTED
% width of the array of memory
        W=N_bl*Pitch_pp;
% length of the array of memory
        L=N_slice*Pitch_pp;
% height of the array of memory
        H=2*Pitch_contact_PT+((N_wl+2)-1)*Pitch_FGT_FGT+2*Pitch_PT_FGT+2*H_contact;
% length of bitline
        L_bl=H;
% length of wordline
        L_wl=W;
% Number of bit of the addresses
        Block_Address= ceil(log2(N_slice));
        Row_Address= ceil(log2(N_wl));
        Column_Address= ceil(log2(N_bl));

%% COMMON PARAMETERS - COMPUTED
% wordline capacitance
        C_wl=Cd_rowpass + C_g_fg * N_bl + C_wl_wire * L_wl;
% bitline capacitance
        C_SA_in=Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n;
        C_bl=2*C_d_pt+N_wl*C_d_fg+C_bl_wire*L_bl+C_SA_in;
        C_SA=Cd_colpass+Cd_slice;
% decoder capacitance
        C_slice_stack=0.5*Cg_sdec_n*Block_Address*N_slice;
        C_row_stack=0.5*Cg_rdec_n*Row_Address*N_wl;
        C_col_stack=0.5*Cg_cdec_n*Column_Address*N_bl;
        Cd_sdec_eval=Block_Address*Cd_sdec_n;
        Cd_rdec_eval=Row_Address*Cd_rdec_n;
        Cd_cdec_eval=Column_Address*Cd_cdec_n;
        C_slice_dec=Cd_sdec_pcharge+Cd_sdec_eval+Cg_sdec_inv_p+Cg_sdec_inv_n;
        C_row_dec=Cd_rdec_pcharge+Cd_rdec_eval+Cg_rdec_inv_p+Cg_rdec_inv_n;
        C_col_dec=Cd_cdec_pcharge+Cd_cdec_eval+Cg_cdec_inv_p+Cg_cdec_inv_n;


%% READ POWER %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

```
117
118 %Slice decoder evaluation stack charge energy
119         E_stack_sdec=0.5*C_slice_stack*(V_on_pt^2);
120
121 % Slice decoder energy
122         E_slice_dec= 0.5*C_slice_dec*(V_on_pt)^2;
123
124 % energy to activate the selected slice
125         E_row_pt= 0.5*(Cg_rowpass*(N_wl+2)+Cg_slice)*V_on_pt^2;
126
127 %Row decoder evaluation stack charge energy
128         E_stack_rdec=0.5*C_row_stack*(V_on_pt^2)*N_slice;
129
130 % Row decoder energy
131         E_row_dec= 0.5*C_row_dec*((V_rd_sel)^2 + ((V_rd_unsel)^2)*(N_wl-1) +
            ↪  2*(V_on_pt^2))*N_slice;
132
133 %Column decoder evaluation stack charge energy
134         E_stack_cdec=0.5*C_col_stack*(V_on_pt^2)*N_slice;
135
136 % Column decoder energy
137         E_col_dec= 0.5*C_col_dec*(V_on_pt)^2*N_slice;
138
139 % dissipated energy in the precharge block
140         E_pre= 0.5*(C_ext_pu_driver+Cg_pre)*(V_bl_prec^2)*N_bl;
141
142 % dissipated energy in the precharge phase BL
143         E_bl= 0.5*C_bl_wire*L_bl*(V_bl_prec^2)*N_bl;
144
145 % energy to switch the selected wordline WL
146         E_sel= 0.5*C_wl*((V_rd_sel)^2);
147
148 % energy to switch the unselected wordlines WL
149         E_unsel= 0.5*C_wl*((V_rd_unsel)^2)*(N_wl-1);
150
151 % energy to read a 1
152         E_1=0.5*C_bl*((V_bl_prec-V_rd_1)^2)*N_bl*(1-p_0);
153
154 % energy to read a 0
155         E_0=0.5*C_bl*((V_bl_prec-V_rd_0)^2)*N_bl*p_0;
156
157 % energy to activate SST and GST
158         E_pt= 2*(0.5*C_g_pt*V_on_pt^2)*N_bl;
```

```
% energy on the string select line and ground select  line
        E_sl= 2*(0.5*(C_ssl_wire*L_wl)* V_on_pt^2);

% energy of the sense amplifier
        E_SA=
        ↪  0.5*C_SA*V_bl_prec*((V_bl_prec-V_rd_0)*p_0+(V_bl_prec-V_rd_1))*(1-p_0)*N_bl;

% energy to read the selected bitline
        E_col_pt= 0.5*Cg_colpass*V_on_pt^2;

%total read energy
        E_read=E_stack_sdec+E_slice_dec+E_row_pt+E_stack_rdec+E_row_dec+E_stack_cdec+E_col_dec+E_pre

% total read power
        P_read=E_read*f_read;           %output of the function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% WRITE POWER
↪   %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Slice decoder evaluation stack charge energy
        E_stack_sdec=0.5*C_slice_stack*(V_on_pt^2);

% Slice decoder energy
        E_slice_dec= 0.5*C_slice_dec*(V_on_pt)^2;

% energy to activate the selected slice
        E_row_pt= 0.5*(Cg_rowpass*(N_wl+2)+Cg_slice)*V_on_pt^2;

%Row decoder evaluation stack charge energy
        E_stack_rdec=0.5*C_row_stack*(V_on_pt^2)*N_slice;

% Row decoder energy
        E_row_dec= 0.5*C_row_dec*((V_prog)^2 + ((V_inhibit)^2)*(N_wl-1) +
        ↪  2*(V_on_pt^2))*N_slice;

% dissipated energy in the precharge block
        E_pre= 0.5*(C_ext_pu_driver+Cg_pre)*(V_bl_prec^2)*N_bl;

% energy to precharge the bitlines
        E_bl= 0.5*C_bl_wire*L_bl*(V_bl_prec^2)*N_bl;
```

```
% energy to switch the selected wordline
        E_sel=0.5*C_wl*(V_prog)^2;

% energy to switch unselected wordlines
        E_unsel=0.5*C_wl*(V_inhibit)^2*(N_wl-1);

% inhibit energy (to mantain
↪  1)
        E_bl_inhibit=0.5*(C_bl-C_d_fg*N_wl)*((0.8*V_inhibit)^2)*N_bl*(1-p_0);

% write energy (to write 0)
        E_bl_sel=(0.5*(C_bl-C_d_fg*N_wl)*(0-V_bl_prec)^2 + E_tunnel)*N_bl*p_0;

% SST/GST energy
        E_pt= 2*(0.5*C_g_pt*V_on_pt^2)*N_bl;

% energy on the string select line and ground select  line
        E_sl= 2*(0.5*(C_ssl_wire*L_wl)* V_on_pt^2);

% total write energy
        E_write=E_stack_sdec+E_slice_dec+E_row_pt+E_stack_rdec+E_row_dec+E_pre+E_bl+E_sel+E_unsel+E_

% total write power
P_write=E_write*f_write;            %output of the function

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% ERASE POWER
↪  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Slice decoder evaluation stack charge energy
        E_stack_sdec=0.5*C_slice_stack*(V_on_pt^2);

% Slice decoder energy
        E_slice_dec= 0.5*C_slice_dec*(V_on_pt)^2;

% energy to activate the selected slice
        E_row_pt= 0.5*(Cg_rowpass*(N_wl+2)+Cg_slice)*V_on_pt^2;

%Row decoder evaluation stack charge energy
        E_stack_rdec=0.5*C_row_stack*(V_on_pt^2)*N_slice;
```

```
% Row decoder energy
        E_row_dec= 0.5*C_row_dec*((V_bl_erase)^2*N_wl+2*(V_on_pt^2))*N_slice;

% dissipated energy in the precharge block
        E_pre= 0.5*(C_ext_pu_driver+Cg_pre)*(V_bl_prec^2)*N_bl;

% energy to precharge the bitline
        E_bl= 0.5*C_bl_wire*L_bl*(V_bl_prec^2)*N_bl;

% energy to start erasing the block
        E_erase_slice=0.5*(C_bl)*((V_bl_erase-V_bl_prec)^2)*N_bl+E_tunnel*N_bl*N_wl;

% SST/GST energy
        E_pt= 2*(0.5*C_g_pt*V_on_pt^2)*N_bl;

% energy on the string select line and ground select  line
        E_sl= 2*(0.5*(C_ssl_wire*L_wl)* V_on_pt^2);

% total erase energy
        E_erase=E_stack_sdec+E_slice_dec+E_row_pt+E_stack_rdec+E_row_dec+E_pre+E_bl+(E_erase_slice+E

% total erase power
P_erase=E_erase*f_erase;            %output of the function

%% DELAY %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%total delay: computed in a read operation (which includes also a precharge operation)
del=0;

%Delay to activate precharge unit - RC delay
    del = (C_ext_pu_driver+Cg_pre)*R_ext_pu_driver + del;

%Precharge delay
    del = ((C_bl_wire*L_bl)*V_bl_prec)/I_on_driver + del;

%Block decoder delay - Gate delay
        Req_n=Req_sdec_n;
        Cd=Cd_sdec_pcharge+Cd_sdec_eval;
        Cl=Cg_sdec_inv_p+Cg_sdec_inv_n;
        Stack_n=Block_Address;
        R_n=Stack_n*Req_n;
    del_block_dec=R_n*(Cd+Cl);
```

```
%Inverter on block decoder's output delay - Gate delay
        Req_p=Req_sdec_inv_p;
        Cd=Cd_sdec_inv_p+Cd_sdec_inv_n;
        Cl=Cg_rowpass*(N_wl+2)+Cg_slice;
        Stack_p=1;
        R_p=Stack_p*Req_p;
        del_block_inv=R_p*(Cd+Cl);

%Row decoder delay - Gate delay
        Req_n=Req_rdec_n;
        Cd=Cd_rdec_pcharge+Cd_rdec_eval;
        Cl=Cg_rdec_inv_p+Cg_rdec_inv_n;
        Stack_n=Row_Address;
        R_n=Stack_n*Req_n;
    del_row_dec=R_n*(Cd+Cl);

%Word line charge delay - Elmore delay
        Req_inv=Req_rdec_inv_p;
        Cd_inv=Cd_rdec_inv_p+Cd_rdec_inv_n;
        Req_pass=Req_rowpass;
        Cd_pass=Cd_rowpass;
        Cl=C_g_pt*N_bl;
        del_row_inv = (Req_inv*(Cd_inv+Cd_pass+Cl)+Req_pass*(Cd_pass+Cl));

%Bit line delay (the K_SA factor is the fraction of the bit line
%delay that influences the total delay, because afterwards the sense
%amplifier is triggered. About 5-10 mV should be the initial differential
%signal for the sense amplifier) - Elmore delay
        k=0;
        tau_eval=0;
        % evaluation delay
        for i=1:length(R)
                k=k+R(i);
                tau_eval=tau_eval+C(i)*k;
        end
        del=K_SA*tau_eval + del;

%Sense amplifier delay

    ↪  del=Req_sa_mod_parallel*(Cd_sa_p+Cd_sa_n+Cg_sa_p+Cg_sa_n+(C_bl_wire*L_bl)+Cd_colpass)
    ↪  + del;

%Column pass transistor and slice transistor delay (after switching of SA) - Elmore
↪  delay
```

```
        Req_pass=Req_colpass;
        Cd_pass=Cd_colpass;
        Cl=0;         %empty load condition
        del = (Req_pass*(Cd_pass+Cd_slice+Cl)+Req_slice*(Cd_slice+Cl)) + del;

if(del_block_dec>del_row_dec)
        del=del_block_dec+del;
else
        del=del_row_dec+del;
end

if(del_block_inv>del_row_inv)
        del=del_block_inv+del;
else
        del=del_row_inv+del;
end

Total_delay=0.69*del;

%% AREA AND VOLUME %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% block decoder
        n_Tr_n_Block_Dec = Block_Address*N_slice+N_slice+Block_Address;
        n_Tr_p_Block_Dec = 2*N_slice+ Block_Address;
        Block_Dec_Area = n_Tr_n_Block_Dec*Tr_n_Area + n_Tr_p_Block_Dec*Tr_p_Area;
% row decoder
        n_Tr_n_Row_Dec = Row_Address*(N_wl+2)+(N_wl+2)+Row_Address;
        n_Tr_p_Row_Dec = 2*(N_wl+2)+ Row_Address;
        Row_Dec_Area = n_Tr_n_Row_Dec*Tr_n_Area +
        ↪  n_Tr_p_Row_Dec*Tr_p_Area;                          %%
% colum decoder
        n_Tr_n_Column_Dec = Column_Address*N_bl+N_bl+Column_Address;
        n_Tr_p_Column_Dec = 2*N_bl+ Column_Address;
        Column_Dec_Area = n_Tr_n_Column_Dec*Tr_n_Area +
        ↪  n_Tr_p_Column_Dec*Tr_p_Area;        %%
%pass_block_row
        n_Tr_n_pass_row_Area =
        ↪  (N_wl+2)*Tr_n_Area;
%pass_block_column post-sense
        n_Tr_n_pass_column_Area =
        ↪  N_bl*Tr_n_Area;                                                    %
%pass_block_slice
        n_Tr_n_pass_slice_Area =
        ↪  N_slice*Tr_n_Area;
```

```
362 %precharge_p_mos
363         n_Tr_p_precharge_Area =
            ↪ N_bl*Tr_p_Area;
364 % sense amplifier
365         n_Tr_n_SA = N_bl*3;
366         n_Tr_p_SA = N_bl*3;
367         SA_Area = n_Tr_n_SA*Tr_n_Area +
            ↪ n_Tr_p_SA*Tr_p_Area;
368
369 %Boundary component area (block decoder, row decoder, colum decoder, sence amplifier,
    ↪ pass transistors, precharge_p_mos)
370         Area_bound_comp = Block_Dec_Area + Row_Dec_Area + Column_Dec_Area + SA_Area +
            ↪ n_Tr_n_pass_row_Area + n_Tr_n_pass_column_Area + n_Tr_n_pass_slice_Area +
            ↪ n_Tr_p_precharge_Area;
371 %Boundary component Volume (block decoder, row decoder, colum decoder, sence
    ↪ amplifier, pass transistors, precharge_p_mos)
372         Volume_bound_comp = Area_bound_comp*L;
373
374 % array area
375         Area_memory=H*W;
376 % array volume
377         Volume_memory=Area_memory*L;
378
379 % total area (total number of transistor * area of single transistor)
380     Total_area = Area_memory +
        ↪ Area_bound_comp;                                  %output of the function
381 % total volume (sum of volumes memory and boundary component)
382         Total_volume = Volume_memory + Volume_bound_comp;                %output of
            ↪ the function
383
384 end
```

Bibliography

[1] V. Mohan, T. Bunker, L. Grupp, S. Gurumurthi, M. R. Stan, S. Swanson (2013), *Modeling Power Consumption of NAND Flash Memories Using FlashPower*, IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN OF INTEGRATED CIRCUITS AND SYSTEMS, VOL. 32, NO. 7, JULY 2013.