Politecnico di Torino

Politecnico di Torino

Master's Degree in Electronic Engineering
Microelectronics Curriculum

Open source Hardware Design: A Register File Case Study

Supervisor:
Prof. M.R. Casu

Candidate:
Vincenzo Giannone

April 2022

Contents

List of Figures

List of Tables

Abstract

This thesis project aims to extend the functionalities of OpenRAM, an open source memory compiler based on Python programming language, capable of generating Static Random Access Memory (SRAM) arrays with adjustable sizes and number of ports by using several technologies, and to characterize the array in terms of power, performance and area about the produced array. The thesis goal consists in enabling OpenRAM to generate and characterize a Register File with 2 read ports and 1 write port by employing a fully open-source design flow and by using SkyWater130 Process Design Kit. The modules needed to build the Register File have been developed in the form of circuit-level and graphic-level descriptions. Then, the source code has been adapted to integrate the new modules and to perform instantiation, placing and routing of them, and to allow the tool to characterize the Register File.

Chapter 1

Introduction

1.1 Open-source design flow

Static Random Access Memories (SRAMs) have become a very important part in many electronic systems, from microprocessors to Systems-on-Chip (SoC) and Application Specific Integrated Circuits (ASICs). The variability of applications leads to the need of multiple SRAM configurations with different number of words, bits per word, columns multiplexing, etc. Of course, this is a problem if all the configurations have to be manually created and for this reason commercial SRAMs are created using SRAM generators.
In many cases, academic IC's design methodologies are limited by the availability of memories because the Process Design Kits (PDKs) are often provided by foundries and vendors, but they do not include any standard cell useful to create memory arrays [1].
Moreover, in some cases researchers cannot use commercial memory generators in their design flow because of academic funding restrictions. Along with this complication, these commercial solutions are limited in customization of the memory sizes and specific components of the memory.
Another issue concerns the possibility to get a view of the back-end features of the contemporary memory compilers. Several companies and foundries have released to their customers some versions of memory compilers that shows the front-end results like timing and power estimations and pins location, but back-end aspects are normally hidden.

This perspective shows the academic environment difficulty about the optimization of memory design and the verification of circuits and methodologies, provoked by the lack of a free automatic tool able to create customizable memory arrays.
This work aims to use an open-source design flow to open the possibility to generate Register Files in SKY 130 nm process node [6].
Building an ASIC requires three different elements, as shown in *Figure 1.1*:

- RTL Design
- EDA tools
- PDK data

As reported in [7], both RTL Design and EDA tools have a quite long history of being open-source. It is possible to mention RISC-V open ISA project for the former and the tools available thanks to the project *Open Circuit Design* for the latter. By having these two parts, RTL Design can be compiled but it is not enough because the PDK is very important to generate the ASIC. It would be not possible to release the GDS (Graphic Design System) of the circuit without it.
SKY 130 nm PDK has been used in this work. It is a collaboration between SkyWater Technology and Google with the aim to provide a complete set of libraries freely. Indeed, SKY 130 nm PDK is open-source, thus it can be downloaded from GitHub at the link:

https://github.com/google/skywater-pdk

There exist other open-source PDKs such as MOSIS 0.5 um, MOSIS 0.35 um and NCSU FreePDK, but in the first two cases the node is too wide, while in the last case the PDK is not manufacturable. SKY 130 nm PDK overcomes these limits since it is a manufacturable technology and, even though the process node is not so recent, it offers good features from performance, power and area point of views yet.

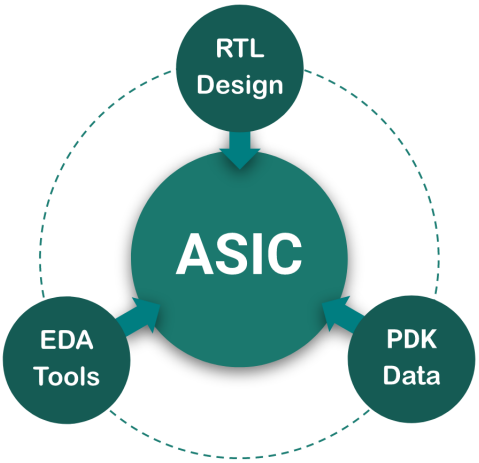


Figure 1.1: ASIC's creation design flow
[7]

1.2 OpenRAM: an open-source memory compiler

The OpenRAM project provides a framework that can generate reference circuit and physical implementation of a SRAMs characterized by any configuration decided by the user concerning size features such as number of words and bits per word, number of ports, number of banks (at most 2) and technology. Technology portability among the ones supported by the tool is a key point. In addition to circuit and layout issues, OpenRAM includes a characterization methodology that supplies timing and power evaluation, so it provides both front-end and back-end results.
The development framework is open-source, therefore it can be freely modified by the user. In this way, anyone can take part to the evolution of this powerful tool by adding new memory configurations or by including the possibility to use a new technology.
OpenRAM is a memory compiler based on object-oriented approach in Python programming language. Source code is available at the link:

https://openram.soe.ucsc.edu/

It works with the aid of other tools to implement design rule check (DRC), layout versus schematic (LVS) and simulation of the memory array just cre-

ated. Thus, there are some dependencies that should be satisfied to ensure a proper working flow. They are listed in the following with the names of some tools mentioned in OpenRAM's official site.

- *Python* 3.5 + with the package *Numpy*
- A SPICE simulator
  - $Ngspice\,26$
  - $HSpice\,I - 2013.12 - 1$
  - $CustomSim\,2017$
  - $Xyce\,7.2$
- A tool able to implement DRC
  - $Magic\,8.3$
  - $Calibre$
- A tool able to implement LVS
  - $Netgen\,1.5$
  - $Calibre$

If any tool for a specific purpose is not installed, then that specific function is disabled with a warning.
By using *Ngspice* for simulation, *Magic* for DRC and *Netgen* for LVS, it is possible to create a completely open-source license-free design flow, as it has been done in this work.
For what concern technology portability, at this moment there are three technology libraries available:

- NCSU FreePDK $45\,nm$
- MOSIS $0.35\,um$ (SCN4M_SUBM)
- SKY $130\,nm$ PDK

1.2.1 Working methodologies

The tool is composed by two different parts, the compiler and the characterizer. They work in combination to provide front-end and back-end results by using the proper methodology. This structure is summarized by the scheme shown in *Figure 1.2*.
Front-end methodology involves the compiler, which requires a manually written input file that configures word size, number of words, number and type of port, number of banks, supply voltage and environmental aspects such as temperature and process corners to be used in the simulation phase to evaluate timing and power results. A library containing all the files regarding technology is also needed to generate the memory array and the information about it can be inserted inside the configuration file by specifying the name of the desired technology among the available ones. Having said that, it is possible to highlight how simple is using OpenRAM to generate SRAM arrays with tunable dimensions and with the desired technology. Configuration file could be useful also to define some options of the tool that will be analyzed later. The compiler provides descriptions of the memory array both at physical and logical level. In particular, the physical front-end results consist of a *GDS* file and a *LEF* (Library Exchange Format) file, which include all the informations about the layout of the memory array. For what concern the logical results, the compiler is able to produce a *Verilog* description of the memory array and a *Spice* netlist that incorporates a circuit-level report about the array itself, including all the sub-modules. Front-end methodology involves also the memory characterizer. It calls a simulator to produce a Liberty file about the estimation of timing and power results based on an analytical model. As reported in the presentation paper of OpenRAM, [1], the characterizer has four main stages: generating the SPICE stimulus, running the circuit simulation, parsing the simulator's output, and producing the characteristics in a Liberty file.

Back-end or characterization methodology requires only the usage of the characterizer that employs a netlist extracted from the *GDSII* layout used to create annotated timing and power results in a Liberty file.
The last procedure provides more accurate outcomes about performance and

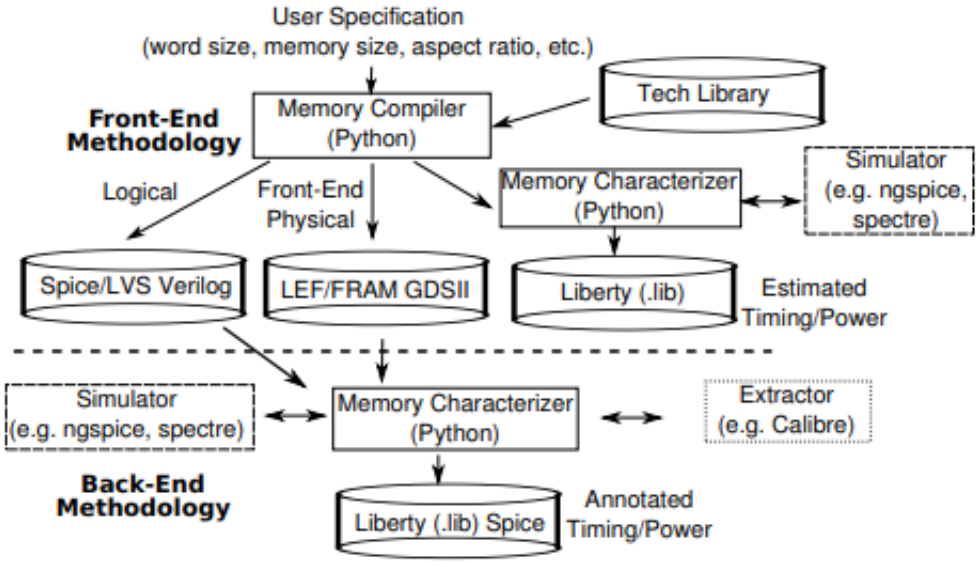


Figure 1.2: Working methodologies of OpenRAM [1].

power with respect to the one involved in front-end methodology. Indeed, the front-end gives only an estimation based on the analytical model, so there are some approximation on the evaluation of the maximum operating frequency and the power dissipated per each cycle. The back-end provides better results through the characterization of the memory just generated, but it could require a long time to be executed, especially if the array has a wide dimension.

One of the options of the tool consists in switching on/off the back-end methodology. It is enough to change the value of the Boolean variable "$analytical\_delay$". If it is set to "$False$", the tool manages to implement the complete characterization of the memory. On the contrary, if it is set to "$True$", then the tool uses the analytical model to estimate timing and power results. Another option allows to choose that model between Elmore model and linear regression through a Boolean variable as just explained.

It is possible to set the options in the specific script "*options.py*" or simply by adding the correspondent variable with the desired value in the configuration file.

1.2.2 Implementation of the tool

The source code is implemented in a hierarchical way, so in the following it will be analyzed with a top-down approach.
The structure is divided into two main directories: compiler and technology. The former contains all the Python scripts to implement the compilation and the characterizaton of the memory array, the latter includes all the technology files and the Python scripts useful to integrate them in OpenRAM.

More in details, technology directory contains a set of sub-directories, one for each PDK available in OpenRAM. Each of them is composed by the models of the hand-made custom cells useful to create the memory array, a library containing *SPICE* models of the devices of that specific technology and a set of files reporting the features of the particular technology.
OpenRAM requires layout (*GDS*) and circuit (*SPICE*) level descriptions of each custom cell employed inside the memory array:

- Bitcell
- Replica bitcell
- Dummy bitcell
- Sense amplifier
- Write driver
- D - Flip Flop
- Tri - gate

Custom cells are created by using a definite technology, thus each technology sub-directory contains different descriptions of the cells.
For what concern devices' *SPICE* models, they are directly transferred from the original PDK without any changing. Models directory accounts for process corners, so it make possible to obtain timing and power results of the memory for each of them by using OpenRAM.
Every technology sub-directory contains a Python file named "*tech.py*", which consists of a declaration of the process layers, the definition of design rules

and the values of the parameters useful to implement the analytical model such as the sizes of the transistors of the custom cells, the resistance per square and the capacitance per square of the metal.
Introducing a new technology involves the creation of a further sub-directory containing all the informations that have been illustrated.

Compiler directory includes all the necessary to create and characterize the memory array. The way the tool is organized consists of a series of classes that provide the basis to instantiate, place and route the modules that have been created through other specific classes. Compiler directory contains also some files such as globals.py and options.py, that are not involved directly on the creation of the array, but they set the conditions in order to make compilation possible.
As reported in [1], the file hierarchy_spice.py contains the *spice* class which has a data structure to maintain the circuit hierarchy. In particular, it sustains the design instances, their pins and their connections. Another important file is hierarchy_layout.py containing *layout* class which includes objects and instances for a module. These two classes provide the base to define the *design* class that, in turn, is the one from which the modules are derived.
Two of the basis classes that worth to mention are *cell* class (custom_cell_properties.py) and *layer* class (custom_layer_properties.py). The former defines all the cells that are used to build the memory array and their pins with the correspondent type, while the latter fixes the layer that are used to create each module.
OpenRAM includes also an integrated custom GDSII library to read, write and modify GDSII files called GdsMill. The *geometry* wrapper class is implemented. It abstracts the GdsMill library and make interfacing easier and porting to other physical layout databeses possible.
The base includes also some other classes that have been created for placing and routing.
About the modules, it is possible to classify them in a hierarchical scheme composed by 3 levels to make explanation clear. By starting from top-level, openram.py script initializes OpenRAM, instantiates a memory design that satisfies all user specifications, performs extraction and characterization and saves all the output files. The class *sram* contains an SRAM design instance,

while the class *sram_base* instantiates the modules that are present inside the memory: control logic, bank and row and column address flip-flops. The *bank* class is responsible for the creation of the single memory bank which consists of 1, 2 or 4 bitcell arrays properly connected with row and column circuitry such as decoders, sense amplifiers, write drivers, column muxes and input/output data flops.
The medium level comprises the classes containing the creation of all the base cells and of all the modules that are made of that cells. For instance, *write_driver* class creates the single write driver and *write_driver_array* class is responsible for tiling the write driver to create the module that will be connected to the bitcell array. Each class places and connects its own sub-circuits while passing its dimensions and port locations up to higher-level modules. At this level it is possible to highligth that not all base cells are custom ones, but some of them are dynamically created by using the lowest level classes to generate modules like decoder and control logic.
OpenRAM provide parametrized transistors and logic gate classes. The classes *ptx*, *pinv*, *nand2* and so on, generate technology-specific layouts so that some modules don't rely on library cells.

1.2.3 Basic usage

Before starting to use OpenRAM it is necessary to define at least three environment variables:

- $OPENRAM\_HOME =$ "/$HOME/*openram/compiler*" pointing to compiler directory
- $OPENRAM\_TECH =$ "/$HOME/*openram/technology*" whose value is the path that brings to technology directory
- $PYTHONPATH =$ "$PYTHONPATH : $OPENRAM_HOME"

Without setting these environment variables the tool gives an error. There are some other environment variables that should be created such as the ones that point to the spice models directory and to the directory that contains the Magic configuration file (if it is used for DRC), but they don't compromise the success of compilation.

Once having fixed environmental variables and completed the configuration file (named "myconfig.py" for instance), it is possible to run OpenRAM in order to generate the memory array. This purpose is reached by executing the following line on terminal:

python3 openram.py myconfig.py

This command must be launched from the compiler directory.
After the time needed for compilation and possibly for characterization, OpenRAM gives a series of files of the kind "*outputname.x*" where x is one of the following extension:

- *.gds* containing the graphical description
- *.lef* including the abstract view of the cells
- *.sp* containing the circuit level description before LVS
- *.html*, a datasheet of the memory about the overall feature such as ports, dimensions, operating conditions and timing and power results obtained through analytical model or characterization
- *.v* with the Verilog description
- *.lvs.sp* containing the circuit level description after LVS
- *.log*, a report of the compilation
- *.lib*, one file for each process corner, containing timing and power results
- *.py* configuration file

OpenRAM gives also three shell scripts, *run_drc.sh*, *run_lvs.sh*, *run_ext.sh*, able to execute rispectively the DRC, the LVS and the parameters extraction of the memory array. They have to be launched by the user after the compilation. Moreover, the tool provides a file named functional_stim.sp, so a spice netlist containing what is needed for a functional test once the period is fixed.

1.3 Work organization

After this introductive part about the open-source design and a brief overview about the way OpenRAM is implemented, in the following parts of the work it will be possible to get into the heart of the Register File project.
The second chapter is dedicated to the analysis of the State of Art about SRAMs and Register Files, with an initial explanation of the basic structure of an SRAM and of the possible solutions commonly used to implement the needed blocks. Then, there will be an overview about the literature's solutions focused on improving one or more features of a Register File.
Afterward, what has been done to allow OpenRAM to generate a multiported SRAM is explained. The work involves both a part related to the hardware design and a part related to the modification of the source code. The third chapter focuses on the design of the blocks needed to build a Register File with 2 read ports and 1 write port from schematic and layout points of view since, as previously said, integrating a module in OpenRAM means providing the tool with a SPICE netlist and a GDS file of the module itself.
The fourth chapter concerns all the needed modifications to OpenRAM's Python framework to give the possibility to instantiate, place and route correctly all the blocks present in the Register File by avoiding design rules violations. Moreover, two of the internal modules of the memory have been created thanks to OpenRAM's framework, so the explanation about the adaptation of the code describing those elements is present in this chapter too.
The fifth chapter contains the results given by the functional simulation of the $2R/1W$ Register File.
Finally, sixth chapter includes conclusion and future developing.

Chapter 2

Register File State of Art

Multiported Register Files (RFs) are critical components in general-purpose microprocessors from performance, area and power points of view. They are used inside superscalar architectures to provide each execution unit with data that are required to be ready as soon as possible in order to stand up to the speed of the processing units themselves. As explained in [4], RFs are required to allow single clock cycle read/write latency to support back-to-back read, read-after-write operations and multiple read/write ports to enable the simultaneous access of several execution units. Moreover, they should be able to store a quantity of data as large as possible to supply the need of execution units, thus avoiding the waste of clock cycles waiting for a data stored in the upper levels of the memory hierarchy. This leads to a wider dimension of the RF and, as a consequence, to a worsening of the performance of the memory due to the larger area. It would be surely possible to satisfy the requirements about area and performance by adopting a particular internal configuration of the RF, but it doesn't come for free. Hence, the consequence of such a changing would be a deterioration of other features, for instance power and cell stability.

It is an example about the way all the parameters which can be used to characterize a memory are associated to each other. It is a continuous trade-off, so the RF should be designed to meet the constraints of the particular application in which it is involved.

In the following, after an introduction about the basic memory configurations

used for single port and multiported SRAMs, there will be an analysis concerning a series of RFs architectures aiming to improve a particular feature.

2.1 Basic structure

SRAMs are volatile memories that lose the information after shutting down the power supply. The basic architecture is shown in Figure 2.1. It consists of the array of cells organized according to $2D$ or $3D$ architecture, the row and column decoders, the column circuitry and the bitline conditioning devices. The cells arranged in the same row share the wordline, while the ones located in the same column share the bitlines. The wordlines are driven by the row decoder and they could be organized in a hierarchical structure with the aim to reduce power consumption and to improve the speed of the memory, as it will be explained in the following. The activation of the desired row allows to perform read and write operations, where the rest of the circuitry comes into play. Hence, before a read operation the bitlines are precharged thanks to the presence of the bitline conditioning circuit. Then, after a row is selected and its cells have sufficiently discharged the bitlines, the bitline sensing circuits placed inside the column circuitry manage to make available the data stored inside the cells of each column located in the activated row to the external. The desired data are driven out of the memory thanks to a series of column muxes that, in turn, are activated by the column decoder. For what concern write operation, after the activation of the desired wordline, the bitlines are driven by the write driver, a component of the column circuitry. Also in this case, the column is selected by means of column decoder. The address received by row and column decoders is provided by the user.
All the modules of the memory, except for the cells, could be enabled by a dedicated signal coming from control logic, a fundamental block that is not present in Figure 2.1. This block handles the internal timing of the memory in order to ensure that all the operations are performed in a proper way.

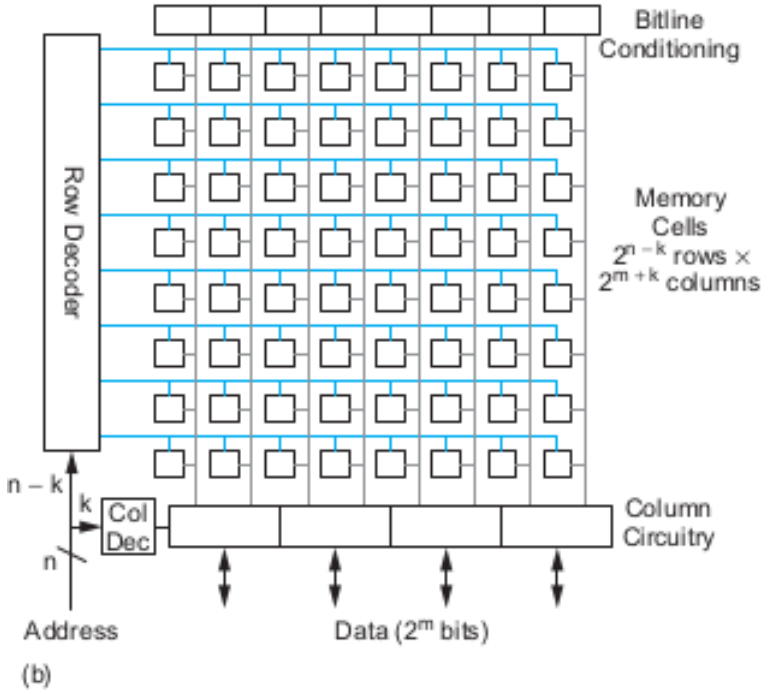


Figure 2.1: Basic standard architecture of a SRAM [2]

.

2.1.1 Basic cells

Cell architectures used in SRAMs can be classified into two groups: single port cells and multiported cells. The former group comprises all the cells that use the same port for both read and write operations and therefore only one at a time of these operations is possible. Instead, the multiported cells allow to perform more than one operation simultaneously thanks to the presence of different read and write ports. Moreover, it is possible to design the cell in order to obtain separate ports for reading and writing.
Multiported cells are used in RFs since they allow parallel accesses from several units in the same cycle, so they guarantee a better performance with respect to the single port cells. Nevertheless, it is necessary to consider that they are more area and power consuming than a single port cell realized with the same technology.
In the following, the basic single port and multi port cells will be analyzed. The part concerning the single port cell contains also some insights about

the cell transistors' sizing and the behaviour of the cell during read and write operations. These aspects are valid also for multi port cells.

6T cell

The $6T$ cell is the most common one used in SRAMs. It is so called because it contains 6 transistors, 4 employed in the data storing latch and 2 that make access possible. The scheme shown in Figure 2.2 highlights the use of a couple of inverters in back-to-back configuration storing the data Q and Q_b, accessible through the pass transistors, one per each side.
The access is possible when the wordline is raised, so when A_1 and A_2 behave as short circuits ideally. At that point it is possible to perform read or write operation by conditioning bitlines properly. Before a reading, it is necessary to precharge both bitlines in order to bring them to a voltage value ideally equal to power supply. Then a row is selected and after a certain amount of time, dependent on the current of the pass transistor and on the bitline capacitance, one of the bitlines is discharged, according to the value of Q. The lines are then connected to the bitline sensing circuitry, so it is possible to read the data stored in the cell. Instead, a data write procedure provides for the precharge of only one bitline, according to the input value that has to be written. If the input is 1, then the line *bit* is brought to the high voltage level, while the complementary bitline *bit_b* must have a low voltage level. When the input data is 0, the bitlines are conditioned in the opposite way.
This configuration establishes differential readings and writings, very relevant characteristic that improves the performance of the entire SRAMs.

The rules for the sizing of the cell's transistors can be deduced by analysing what happens during read and write processes.
Figure 2.3 contains the qualitative waveforms of the signals involved in a read operation of a 0 stored in a $6T$ cell. When wordline is at the high level both $D1$ and $A1$ can conduct charges, so the bitline *bit* is discharged through the series of these two transistors that behave like resistances. This phenomenon creates a voltage divider between $D1$ and $A1$, so the voltage at the node Q rises. If it reaches an high enough value, $D2$ could switch on with the subsequent discharging of the complementary bitline *bit_b* and of the node

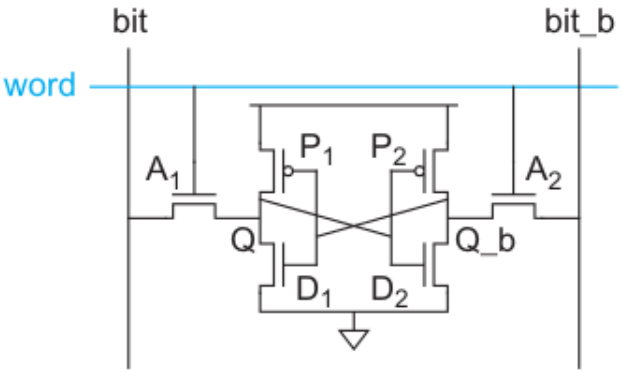


Figure 2.2: 6T cell's schematic. [2]

Q_b. This means altering the cell content with the result to make it not valid anymore. The problem can be avoided by sizing $D1$ and $D2$ to make sure that Q doesn't reach a too great voltage. In particular, $D1$ must be wider than $A1$ in order to show a lower resistance, so most of the voltage drops on $A1$. For the same reason $D2$ must be wider than $A2$. In this way, reading operations are non destructive.

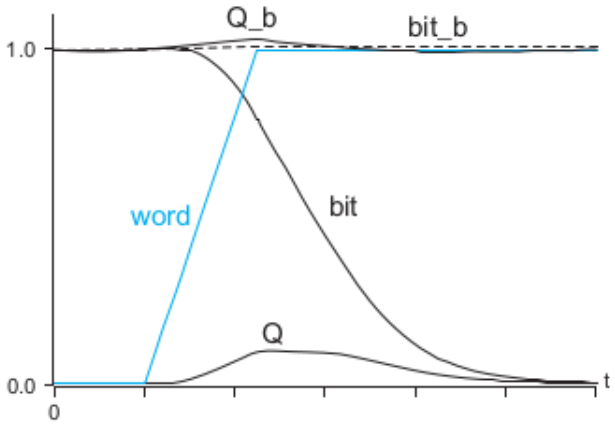


Figure 2.3: Read operation in a 6T cell. The labels refer to Figure 2.2. [2]

Figure 2.4 shows the qualitative waveforms implicated in a write operation of a 1, so bitline *bit* has to be charged, while complementary bitline has to be discharged. When row decoder drives the wordline stimulus, the voltage at the node Q increases slowly in the initial part, then there is a variation of slope which causes the speed up of the rise. Slope increasing is due to the

rising of Q itself since it is the input of the inverter composed by $D2$ and $P2$. $D2$ switches on, so the node Q_b is discharged and Q is forced to 1 beacuse of the turning on of $P1$. This mechanism occurs only if the access transistor is wider than the pull up transistor of the related inverter, so in this example $A1$ and $A2$ must be respectively larger than $P1$ and $P2$.
In summary, the nMOS pull down transistor of the cross-coupled inverters must be strong, the access transistors must be of intermediate strength and the pMOS pull up transistor must be weak [2].

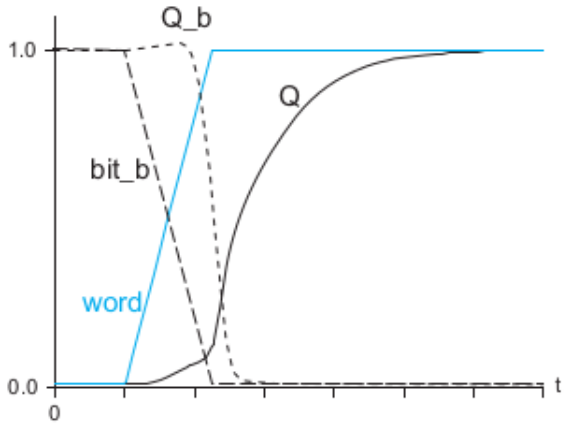


Figure 2.4: Write operation in a 6T cell. The labels refer to Figure 2.2. [2]

Sizing has a fundamental effect on cell stability, very important aspect that must be considered during the design. Stability is evaluated through a parameter called *Static Noise Margin* (SNM), that takes a different definition according to the situation in which the cell is tested. In general, SNM is the maximum noise level that can be applied to the cell to keep the data intact. If noise level is higher than SNM, data will be lost.
SNM can assume three interpretations:

- Hold margin. The stored data could be lost even without any read/write operation. Hold margin is estimated thanks to a simulation based on the circuit in Figure 2.5, where V_n is a voltage source that assumes the role of noise generator. When it is off, the cross-coupled inverters are characterized by the so called *butterfly diagram* in Figure 2.6 which is not perfectly symmetrical because of the LO skewed inverters.

When noise sources switch on and V_n has a positive value, curve I is moved to the left, while curve II is shifted up [2]. At a certain point noise reaches an enough high level, so it eliminates the stable states represented by the two points with coordinates $V_1 = V_{DD}, V_2 = 0$ and $V_1 = 0, V_2 = V_{DD}$. This would mean the loss of the stored data.
SNM is determined by the length of the sides of the maximum area squares that can be inscribed inside the two curves. If the two squares are not identical, the SNM is the lesser of the two sides.
SNM increases with power supply and transistors' threshold voltage.

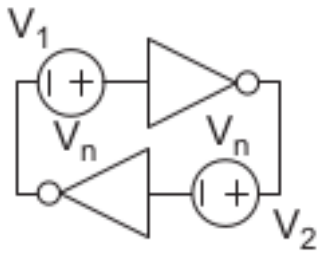


Figure 2.5: Circuit used for the evaluation of hold margin [2].

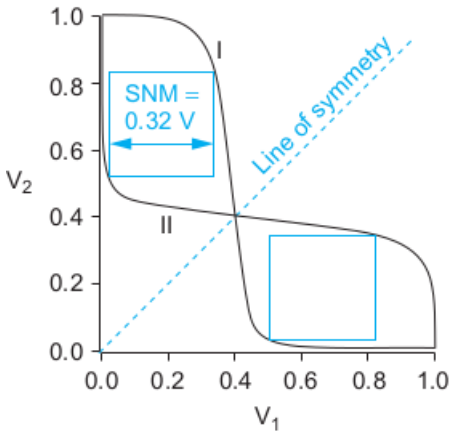


Figure 2.6: Characteristic of the circuit in Figure 2.5 [2].

- Read margin. Read operation can be distructive if the cell's transistors are not sized properly. This situation can be tested by using the circuit in Figure 2.7 with the access transistors diode connected in order to simulate the two precharged bitlines. From Figure 2.8 can be seen

that V_2 is not able to reach $0\,V$ because a conflict occurs between the upper inverter and the right access transistor when V_2 has to assume a low level voltage. Moreover, it is possible to notice that the zone of inscription of the square is smaller than the one in Figure 2.6, so read margin is lower than hold margin. When noise increases, the curves move away from each other until there is no stable state and the data is destroyed.

Read margin can be increased in two ways. The first one is focused on sizing of the transistors of the cell, the second one provides for using a lower voltage to drive the wordline with respect to V_{DD} in order to decrease the strenght of the pass transistors. The last solution is rarely used because having the same voltage level for bitlines and wordlines is more convenient.

The application of the first method allows to find a design equation. By considering a situation in which $Q = 0$, the higher is the voltage level reached by Q during a read operation, the higher is the probability to corrupt the data. In that case pull up pMOS $P1$ is switched off, so $A1$ and $D1$ are passed through by the same current. I_{A1} and I_{D1} can be set equal by noticing that $D1$ is in triode region, while $A1$ works in saturation condition. By marking as r the ratio between the width of the pull down nMOS and the width of the correspondent access transistor, as V_{DSATn} the value of the voltage between drain and source of a nMOS that makes the transistor enters saturation region and as V_{tn} the threshold voltage of a nMOS (considered constant), by supposing the long channel model as valid, it is possible to write:

$$\beta_{D1}((V_{DD} - V_{tn})V_Q - \frac{V_Q^2}{2}) = \beta_{A1}((V_{DD} - V_Q - V_{tn})V_{DSATn} - \frac{V_{DSATn}^2}{2})$$

where β is equal to:

$$\beta = \frac{\mu_n C_{ox}}{2}\frac{W}{L}$$

By setting $r = \frac{\beta_{D1}}{\beta_{A1}}$, so proportional to the ratio of the widths:

$$V_Q = \frac{1}{r}(V_{DSATn} + r(V_{DD} - V_{tn}) - \sqrt{V_{DSATn}^2(1+r) + r^2(V_{DD} - V_{tn})^2})$$

V_Q does not rise over a certain level that is usually fixed to V_{tn} in order to avoid the switching on of $D2$ (or $D1$, it depends on the side in which the 0 is stored). In this way it is possible to find the minimum r for which a read operation is non destructive.

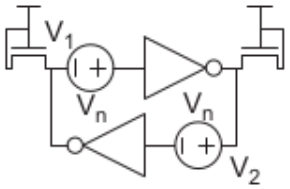


Figure 2.7: Circuit used for the evaluation of read margin [2].

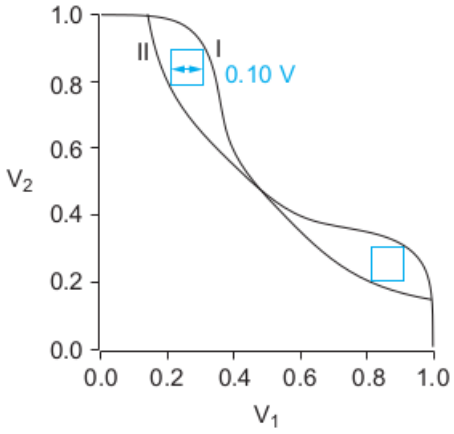


Figure 2.8: Characteristic of the circuit in Figure 2.7 [2].

- Write margin. A writing operation could fail because of, as it is said previously, a not correct sizing of access transistor and pull up pMOS. The circuit used to estimate write margin, or writability of the cell, is shown in Figure 2.9, where one access transistor is diode connected, while the other one has a grounded terminale to simulate the high and low voltage level bitline respectively. The resulting graph, valid in static conditions, so when $V_n = 0$, is displayed in Figure 2.10, which appears distorted with respect to the *butterfly diagram* in Figure 2.6 because of the conflict on node V_1 between the access transistor that forces a 0 and the pull up of the lower inverter that forces a 1. Moreover, there is

only one stable state that occurs for $V_1 = 0$ and $V_2 = V_{DD}$ because in the scheme V_1 is connected to the grounded pass transistor, while V_2 to the diode connected transistor. When V_n increases, the curves move until there exists another stable state that prevents the writing.
Also the write margin can be enlarged in two ways: operating on the sizing and increasing wordline high voltage level. Notice that read margin is inversely proportional to the voltage level of the wordline when it is on, so it is possible to state that read margin and write margin are in contrast.
For what concern sizing, write margin depends on p, defined as the ratio between the width of the pull up pMOS and the width of the access transistor. Write margin improves if p decreases, so when the width of the access transistor increases. This is another point to highlight the contrast between read and write margin. By considering a situation in which a 0 has to be written, during the operation $P1$ and $A1$ are on, while $D1$ is off, so they are passed through by the same current. By setting equal I_{A1} and I_{P1} it is possible to obtain another design equation:

$$V_Q = (V_{DD} - V_{tn}) - \sqrt{(V_{DD} - V_{tn})^2 - 2\frac{\mu_p}{\mu_n} p((V_{DD} - |V_{tp}|)V_{DSATp} - \frac{V_{DSATp}^2}{2})}$$

p can be expressed as:

$$p = \frac{\beta_{P1}}{\beta_{A1}} = \frac{\mu_p W_{P1} L_{A1}}{\mu_n W_{A1} L_{P1}}$$

Its value can be tuned by playing on the channel length of the pMOS, that in SRAM is sometimes fixed to 3λ with all the related reliability issues compensated by redundancy.
p is evaluated by fixing a value of V_Q that depends on the input data. If it is 0, as in the just analysed case, V_Q should not rise over a certain value, usually the threshold voltage of $D2$, in order to avoid the discharging of the node Q_b.

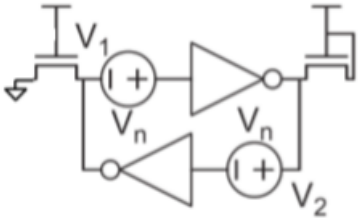


Figure 2.9: Circuit used for the evaluation of write margin.

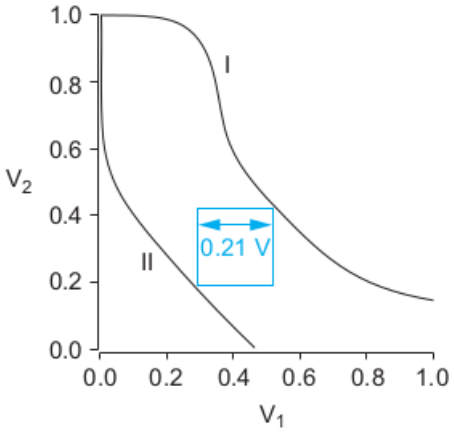


Figure 2.10: Characteristic of the circuit in Figure 2.9 [2].

8T cell

The $8T$ cell is composed by 8 transistors arranged according to the scheme of Figure 2.11. It follows the topology of $6T$ cell but it provides for separated bitlines and wordlines for read and write procedures. Writing a data requires the activation of write wordline and of one of the write bitlines according to the input data, so it is the same of $6T$ cell. What changes is reading. Hence, read operation is no more differential, the output data depends on the activation of a nMOS driven by the stored data. Read wordline is activated and if $Q = 1$, then read bitline can be discharged through the path offered by the two transistors on the right. When $Q = 0$ read bitline keeps an high voltage value. Notice that at the end of reading cycle, read bitline shows the opposite value with respect to stored data, so it should be processed in the proper way.

During reading, node Q voltage has not been altered in any way, so it is

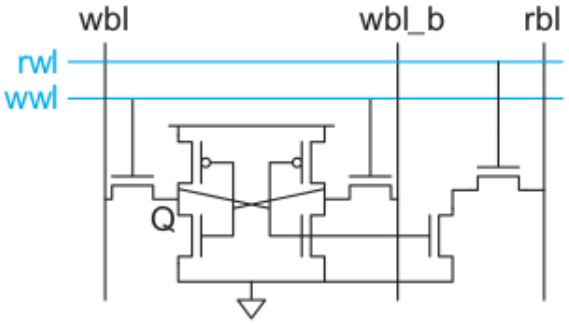


Figure 2.11: Schematic of 8T cell [2].

possible to conclude that 8T cell guarantees an optimal read stability. In other words read margin is equal to hold margin.
Moreover, this cell can be seen as a first example of multi ported architecture because it has different lines for read and write operations.

12T cell

The 12T cell can be used in order to further improve memory cell's robustness. As it is possible to see from Figure 2.12, this cell includes a latch and a tristate port used during reading. Write ad read wordlines are separated but bitline is common.
Bitline doesn't need to be precharged before reading because tristate port is able to drive it in any case. This feature is very relevant, leakage power and noise are reduced since there are no precharged floating nodes. Of course, the cost for these is the larger area of the cell, which results in a low density of the SRAM array.

6T multi ported cell

This cell is the same of the 6T previously analysed except for the presence of two wordlines, one per each access transistor. If control manages to provide signals by following a particular timing, this cell allows the user to perform two reads and one write in the same cycle, as commonly required for a Register File in a single-issue RISC processor [2]. In that case, read operation would become single ended, while write procedure would remain differential.

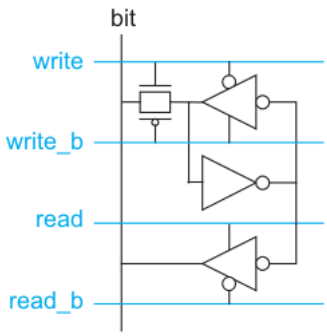


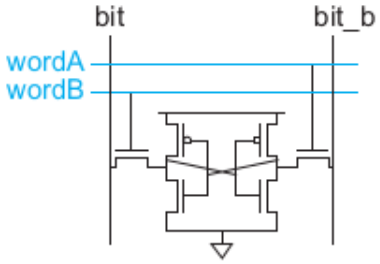
Figure 2.12: Schematic of 12T cell [2].

The technique that should be used is called *time multiplexing* or *double pumping*. It consists in reading in the first half of the cycle and writing in the other half.
There will be some examples about the use of this kind of method in the following section.

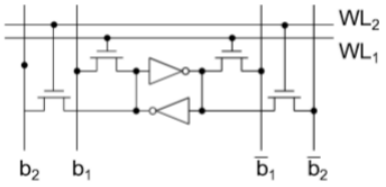
Dual ported SRAM cell

As shown in Figure 2.13b, this cell is formed by a latch and 4 access transistors that guarantee the possibility to get two read/write ports. It works exactly as the 6T cell but read and write stabilities are critical. Read operation disturb is described in [8], it arises when two bitcells in the same row are accessed. Both wordlines are activated and by focusing on a bitcell that performs a read operation on one port, the other port operates a dummy reading. If a 0 is stored on that bitcell, the bitline should be discharged, but it is hampered by the other bitline which is precharged to 1.
Concerning write stability, as explained in [9], when both ports are set up in order to perform a read/write operation in the same row the write-disturb issue arises. By considering a bitcell in which the input value has to be written, both wordlines are activated but one couple of bitlines is set for a dummy reading, the other couple for writing. The discharge of the storage node would be difficult because of the precharged bitline set for reading and connected to storage node itself that hampers the data flip. In the case of different rows access the write disturbance is not a problem.
This kind of cell could be useful in the implementation of *FIFO* queues which

allow to read and write data with a different rate. Dual port cells can be used in that case since there is the need to read and write data simultaneously and by operating in the same cell if the queue is empty.



(a) Schematic of $6T$ multi ported cell [2].



(b) Schematic of dual ported SRAM cell.

Figure 2.13: Schematics of $6T$ multi ported cell and dual ported SRAM cell.

Register File classical cell

RFs require cells able to provide the possibility of doing more than one operation in the same cycle through multiple ports, as it has been said at the beginning of this chapter. The architecture of such a cell has to be slightly different with respect to the one of a single port cell because of some aspects linked to stability and area.
Reading should be a single ended procedure in order to save area, but at the same time each read port should be isolated by the stored data to get a reasonable read margin. A single ended read port is obtained at the cost of two series transistors, one bitline and one wordline. A differential read port would double the cost.
For what concerns writing, it should be kept a differential operation in order to avoid a too low writability, so each write port would cost two bitlines, a wordline and two access transistors. Actually, it is better to prevent the use of extra lines since in cells with many ports, the area of the wires dominates the area of the transistors [2].
This line of reasoning brings to adopt a cell architecture like the one in Figure 2.14. Read port works as explained in 2.1.1 since it has the same structure. There is a single write bitline that is inverted to ensure differential writings.

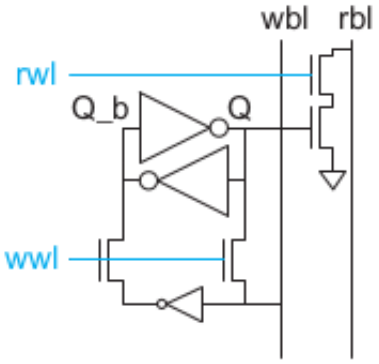


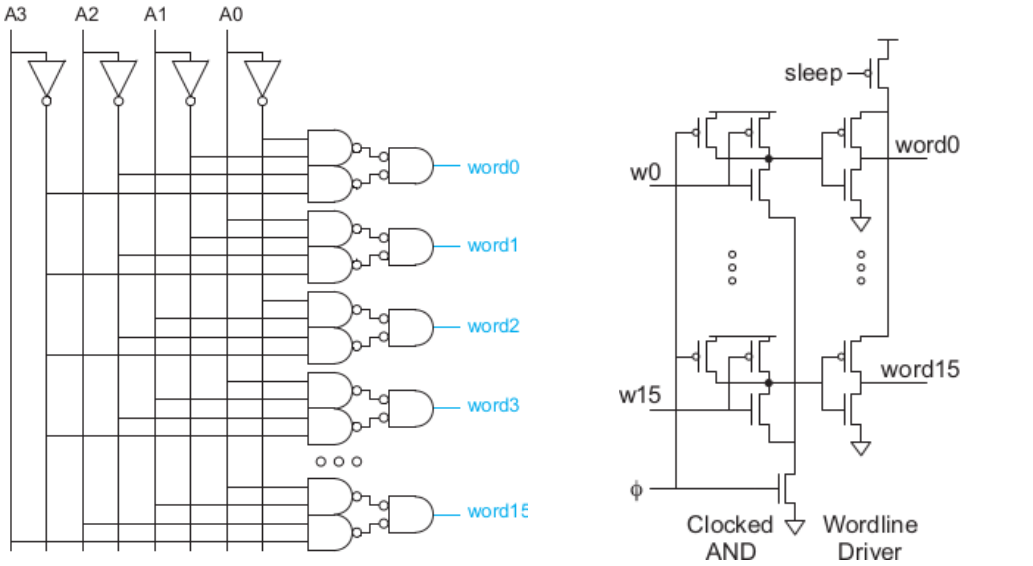
Figure 2.14: Schematic of classical Register File cell [2].

2.1.2 Row circuitry

Row circuitry is a module that provides wordline signals connected directly to access transistors of the cells. It consists of a decoder whose output are connected to drivers, one per each wordline.
Figure 2.15a shows a possible implementation of a $4 : 16$ decoder based on a pair of 2-inputs NAND gates followed by a 2-input NOR gate.
Figure 2.15b contains the circuit-level representation of a wordline driver with 16 words. Each word coming from row decoder must be synchronized with clock signal to obtain a proper bitline timing. In wordline driver, all the words are collected and put in logic AND with a signal coordinated with clock and provided by control logic. This implementation involves also a *sleep* signal connected to the gate of a pMOS in order to implement *power gating* technique, which consists in forcing *sleep* to 1 when wordline driver doesn't need to work with the aim to save leakage power since power supply is ideally disconnected.
A key point to highlight consists in matching the pitch of the row decoder and of the wordline driver with the pitch of the cell. It could be very tricky when a small cell is used in the array.

Predecoding

In Figure 2.15a implementation there is a wastefullness of connections because many NAND gates share the same inputs. The problem can be solved



(a) 4:16 row decoder [2].

(b) 16-words wordline driver [2].

Figure 2.15: Schematics of 4:16 row decoder and 16-words wordline driver.

by applying *predecoding* technique, which consists in factoring NANDs' outputs [2]. An example of a $4 : 16$ decoder realized by using predecoding is shown in Figure 2.16, where the number of ports is higher, so electrical effort is greater, but overall path effort is the same because branching effort is reduced.

Hierarchical wordlines

The wordline is highly loaded since it enables read or write or both operations for each cell in an array row. If the array is large, delay can be very high due to RC effects by taking into account also the fact that lower levels of metal are used to build wordlines. [2]
One possible solution consists in dividing the single wordline in order to arrange it in a hierarchical way, with global and local wordlines. The example is displayed in Figure 2.17.
A local wordline is relatively short and it drives a small group of cells, while global wordline is still long but it is made with upper-level of metals and it is less loaded.
This technique also ensures a lower power consumption because only bitlines

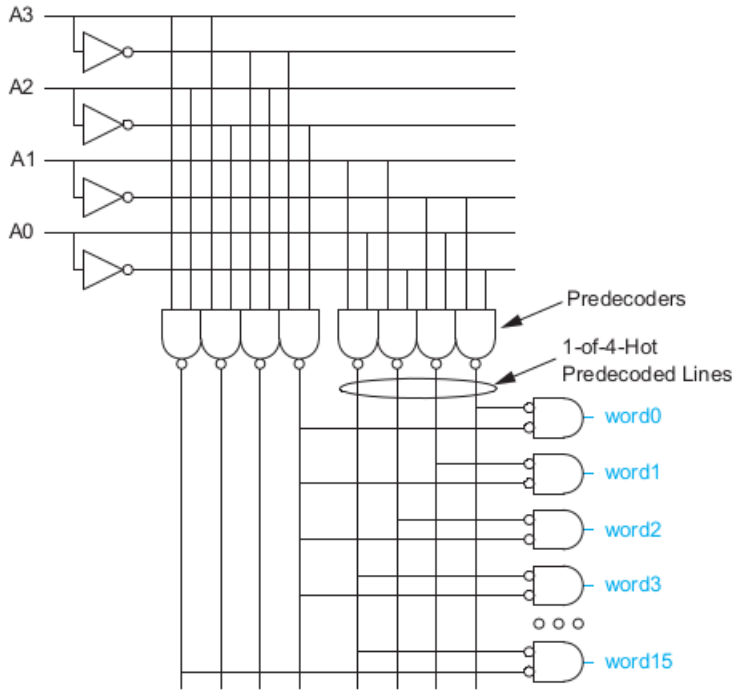


Figure 2.16: 4:16 row decoder with predecoding technique [2].

activated by the local wordline will switch.

2.1.3 Column circuitry

Column circuitry consists of the modules needed to manage bitlines during read and write procedures. Bitline conditioning circuit, bitline sensing circuit, write driver and column multiplexer are inserted in the memory in a proper way with the aim to allow the interaction between the external and the storing data cells.

Bitline sensing

A proper sensing of bitline value after a read operation is fundamental to ensure good performance during reading and a very low error probability. According to the circuit used to perform sensing, it can be divided in two categories: large-signal and small-signal sensing.
The former is called also single-ended sensing and it provides for a bitline

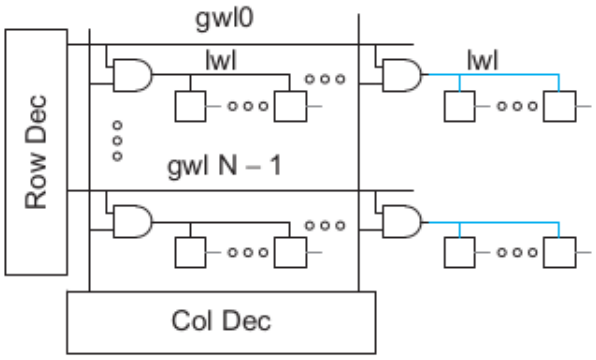


Figure 2.17: Hierarchical wordlines [2].

swing from V_{DD} to 0. The bitline drives a circuit that could be, for instance, an high-skewed inverter properly sized in order to accelerate the procedure. The delay is proportional to the number of words attached to the bitline, so if the array is large, then a single inverter could be not enough to ensure an appropriate reading speed. A method to solve that problem consists in adopting a hierarchical scheme for bitlines, with global and local lines as shown in Figure 2.18. Each local bitline is connected to a small group of cells and, in combination with another local bitline, can pull down the global bitline, which drives an high-skewed inverter. Local bitline can be seen as an unfooted domino multiplexer, while global bitline is an unfooted domino OR gate.

The constraints about the using of hierarchical bitline scheme are linked to leakage power and delay. A dynamic multiplexer has a constant logical effort, but delay is proportional to the number of inputs, so, in order to keep acceptable performance, a single local bitline cannot be attached to more than 32 words [2]. Moreover, the number of transistors connected to a local bitline is limited by the fact that it could be discharged because of leakage. The worst case is the one in which the cell being read contains a 0 and all the others a 1. Local bitline should keep the 1 value, but subthreshold current tends to pull it down [2].

Small-signal sensing is also called differential sensing, it uses a sense amplifier to read the data contained in one cell. Each sense amplifier is driven by a bitline and its complementary. In this case, the bitlines swing by a small amount

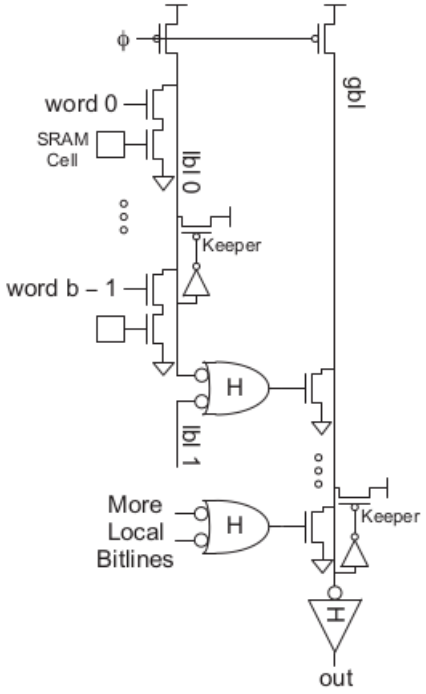


Figure 2.18: Hierarchical bitline with large signal sensing [2].

(about $100-300\,mV$), then, when sensing is completed, wordline must be deactivated in order to prevent a useless extra bitlines voltage variation to save power consumption. Differential sensing is faster than single-ended sensing but it is more area consuming.

Figure 2.19 shows a first example of sense amplifier. It contains a differential pair and a current mirror to balance current in the two branches. Current is provided by M_5, activated by a signal coming from control logic. The output of differential amplifier is then inverted because it is an inverting stage.

Another implementation can be viewed in Figure 2.20a, the scheme consists of a latch that implements a *rigenerative feedback*, two pMOS labeled as *isolation transistors* and a nMOS that provide current to the latch and that is driven by a clocked signal. When the clocked signal turns on the bottom nMOS, sense amplifier starts working and one of the output signals is pulled down, while the other one is pulled up through the feedback. The isolations transistors are useful to speed up the response and to save power by disconnecting the outputs from the capacitive load of the entire bitline during

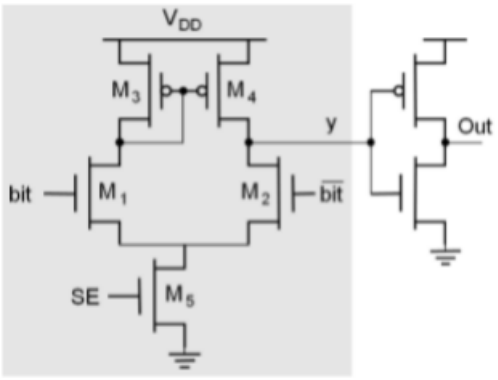
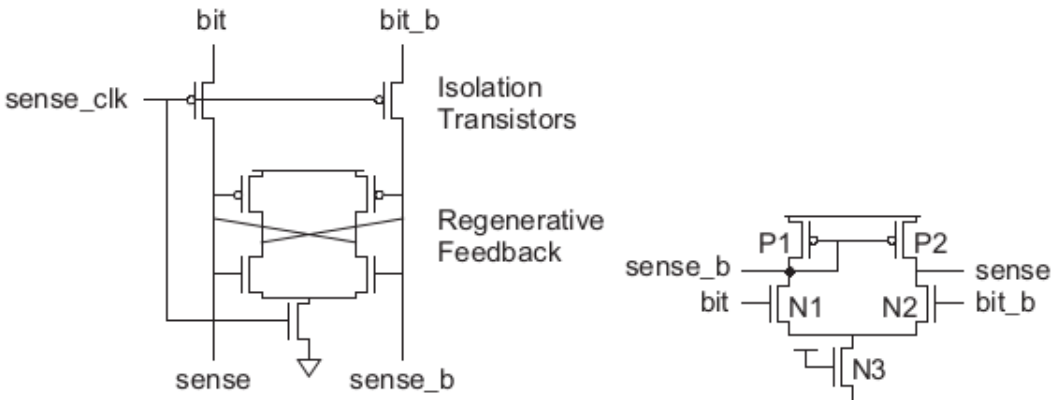


Figure 2.19: Differential amplifier used to sense bitlines.

sensing.

The last implementation of sense amplifier is present in Figure 2.20b and it is composed by a differential pair, a current mirror and a nMOS that works as a current sink. The difference with respect to the one in Figure 2.19 is the lack of any clocked signal to enable sensing. The current sink is gate-connected directly to power supply, so this kind of circuit is always on and it implies a significant power consumption.



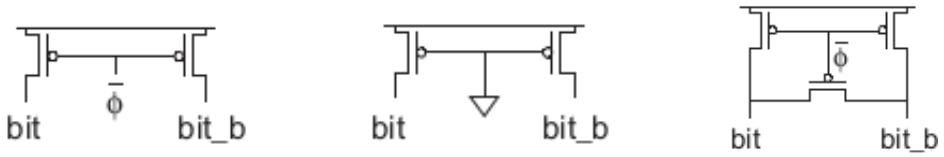
(a) Sense amplifier with regenerative feedback [2].

(b) Sense amplifier without enable signal [2].

Figure 2.20: Possible implementations of a sense amplifier.

Bitline conditioning

Before reading and writing, bitlines must be precharged as explained previously by a dedicated circuit that constitutes the bitline conditioning module. The first example is shown in Figure 2.21a, it is simply composed by two pMOS driven by a clocked signal provided by control logic.
Another implementation in Figure 2.21b doesn't require any enable signal but it is done in pseudo-nMOS logic. In this case the pMOS must be weak to reduce the consequences of the contention between pull up and pull down represented by the transistors of the cell that offer a path to discharge the bitline. This solution slows the read operation and it cannot be used in low-voltage SRAMs [2].
Last example is shown in Figure 2.21c. It contains three pMOS with the same enable signal. The additive transistor is used to obtain well balanced charge in the two bitlines, so to avoid that the two lines have a different voltage after precharge due to a different residual charge as a result of the previous reading.



(a) (b) (c)

Figure 2.21: Bitline conditioning circuits [2].

Enable sensing signal providing

Some bitline sensing circuits require a signal to be enabled. That signal is provided by control logic, but it is very important to supply it with the right timing to match the delay of row decoder, wordline and bitline. The most straightforward solution uses an inverter chain that introduces a fixed amount of delay. It is not the best possible solution because it doesn't track the delay across environmental and process variations since the feature of the transistors can be different from one point to another of the SRAM.
A worthy solution is based on *replica bitline technique* [10]. It requires the

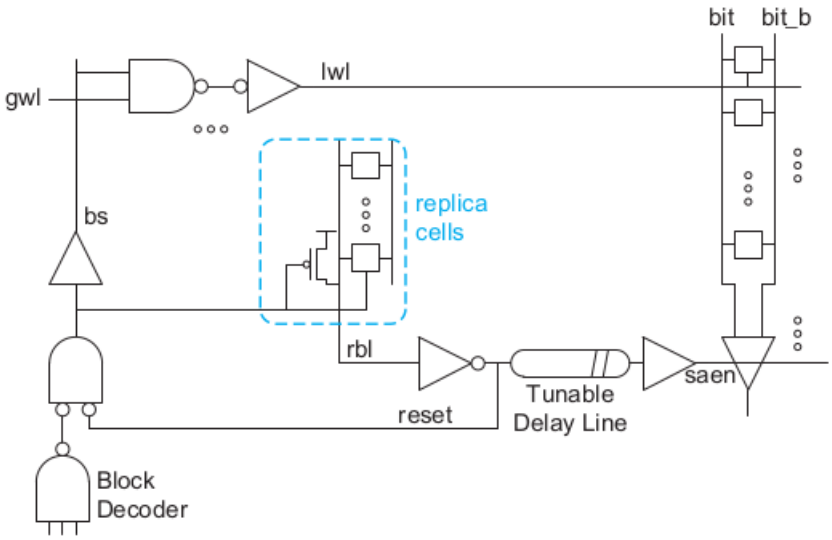


Figure 2.22: Replica bitline implementation [2].

presence of an additive column containing a bitline and *replica cells*, that are identical to the ones used inside the array to store data, but they have Q forced to 0 because replica bitline must be discharged. This technique is implemented in the scheme present in Figure 2.22, where the decoder gives a signal, *bs*, which selects a memory block, so the proper local wordline is activated and the cell's data is read. At the same time, the signal given by block decoder activates one cell of the replica column which incorporates a number of cells r times lower than an array column, so replica bitline discharges about r times faster than a normal column. When *rbl* is brought to a low voltage level, *reset* is generated and it causes the deactivation of the wordline by blocking the discharging one of the bitlines to $\frac{V_{DD}}{r}$, so r can be tuned to match the desired bitline swing. Replica path involves most of the elements of a real path, so in this way the delay tracks very well the process and e environmental variations. Delay can be tuned if variation is greater than expected or if the circuits does not work properly through a stage that includes some fuses.

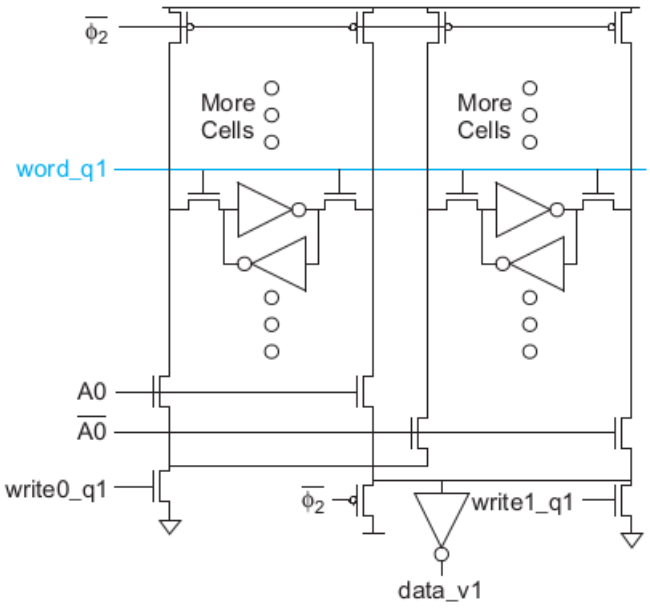


Figure 2.23: Column multiplexing [2].

Column multiplexing

Column multiplexing is a technique used in SRAMs to allow a read or write operation of a particular word inside a row. If a row contains m words of k bits, a $2^k : 1$ column multiplexer is required to operate only on a single word. It is useful to reduce the number of signal sensing circuits and write drivers required inside the SRAMs since column pitch is quite narrow, so it could be challenging to insert one of them per each column from layout and area point of view. The reduction of the number of column circuits leads also to a decreasing of power dissipation. For what concern the influence on performance, column decoding is done in parallel to row decoding, so it does not impact on critical path [2].
Figure 2.23 shows a two-way column multiplexer based on nMOS pass transistors. Large signal sensing circuits and write drivers are both connected to the output of the multiplexer, that is precharged high through a pMOS.

2.2 Analysis of Register File State of Art

There exist four different main parameters that can be considered to quantify the effectiveness of a Register File:

- Performance in terms of maximum operating frequency and memory bandwidth.
- Power, both static and dynamic contributions.
- Area, very important feature since the cost of the integrated circuits grows as the fourth power of area [11].
- Stability in terms of Static Noise Margin.

As previously told, there is a trade off between those features, so the enhancement of one of them provokes the worsening of the others. In the following, some architectures used inside RFs will be described by dividing them according to the particular feature improved in the design.

2.2.1 Performance

Time multiplexing or *double pumping* is one of the techniques useful to improve the performance of a memory. It requires a very accurate timing of the control signals to perform reading in the first half of the cycle and writing in the other half.
An application of this method is explained in [3], where the design of a RF with 6 read ports and 6 write ports is described. Bitcell's schematic is shown in Figure 2.24 and it is possible to notice that it contains only 3 ports for reading and 3 ports for writing. Double pumping read and write operations allows to obtain 6 reads and 6 writes by avoiding the increasing in area, power and latency due to the replication of the bitcell or to the addition of further ports [3].
Another important aspect to report is the using of hierarchical bitline techinque described in Section 2.1.3.
The RF is organized in four banks of 32 words with 32 bits per wordline (WL) and 16 bits per local read bitline (LRBL). The four banks share the

global read bitline (GRBL). RF read circuit consists of decode logic, LRBL precharge and evaluation and GRBL. The key point that makes possible the implementation of time multiplexing is duplicating decode logic and GRBL. In this way, it is possible to perform two decode and GRBL operations per cycle while only sharing LRBL, so the sequential operations during a clock cycle are only two LRBL precharges and evaluations, one for each reading (r0 and r1). RF write circuit has the same structure of the RF read circuit, but only decoding circuitry is duplicated. Moreover, LWBL is shared by all the 32 cells in a row.

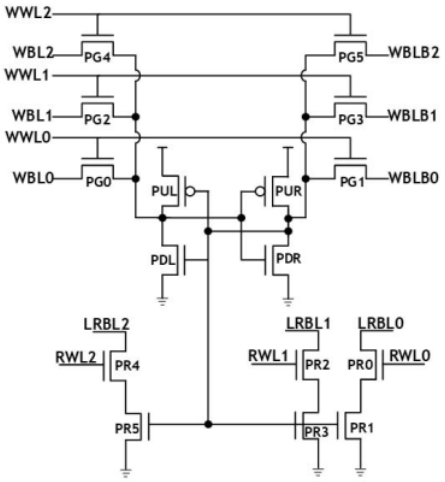


Figure 2.24: 3R3W bitcell schematic [3].

Figure 2.25 shows the scheme used to generate clock signals for two parallel operations $clk0$ and $clk1$, the two addresses and the two bank clocks. $clk0$ and $clk1$ are generated thanks to the clock pulse generator (CPG) circuit driven by global clock $clkg$. The value of the output of CPG is controlled by the rst_clk signal in double pumping mode, otherwise it follows $clkg$.
A key point to achieve double pumping is the presence of read/write replica circuits (RRC/WRC), which use the replica bitline technique explained in Section 2.1.3 to provide a proper clock-pulse width in order to allow the performing of two read/write operations per cycle. Hence, when the signal *done* is asserted and time multiplexing is enabled, then $clk0$ and $clk1$ pulse widths are shaped. These two signal are also the inputs of two different clock gating cells (CGC) that generate the bank clocks.

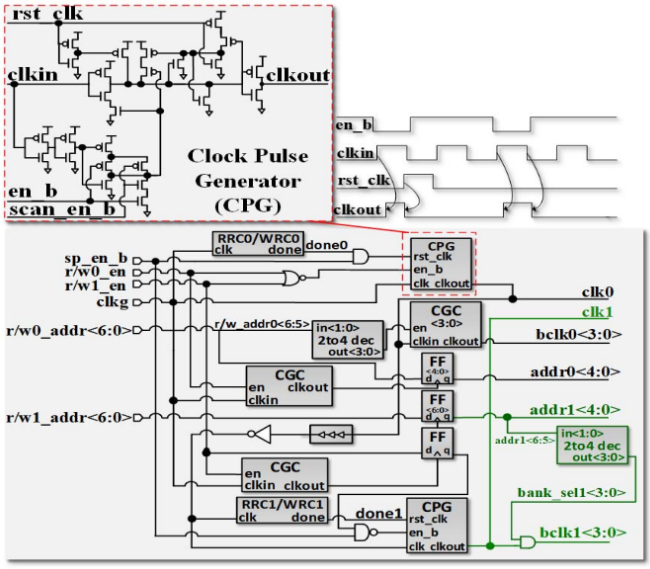


Figure 2.25: Decoding circuit for double pumped read and write [3].

Figure 2.26 offers a view of the hierarchical bitline arrangement for read operation. There are two LRBLs per each row since each of them is connected to 16 cells and there are 32 cells per row. LRBLs are put in logic NAND to provide the input of the GRBLs, that are shared by the four banks.

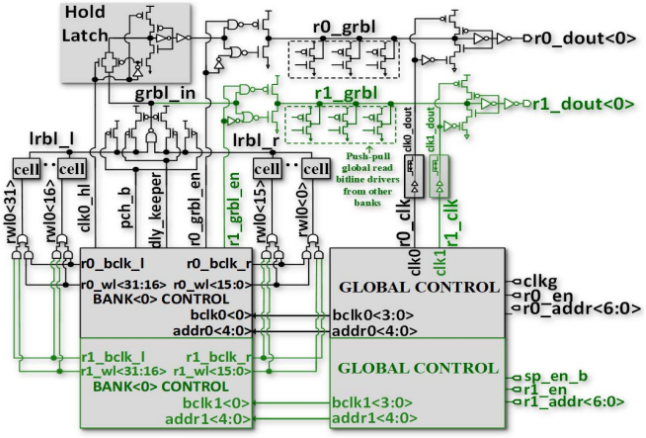


Figure 2.26: Circuit used for double pumped read [3].

Write circuit is shown in Figure 2.27. As previously said, only decoding circuitry is duplicated, while GWBL and LWBL are shared. It is possible because write operation is less timing critical than read operation. Write

decode is nearly identical to read decode except for the replica circuit used to sharp $clk0$ and $clk1$. Input data are collected by a write data flip flop that provides the GWBL signal, transferred on LWBL thanks to a local write bitline driver.

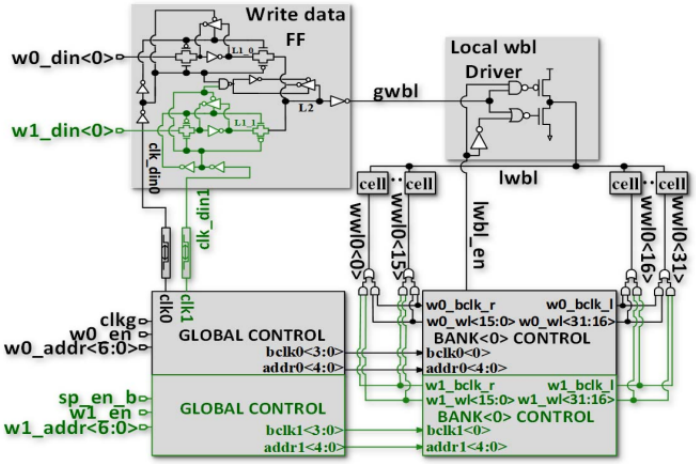


Figure 2.27: Circuit used for double pumped write [3].

Implementing double-pumping technique by using this strategy leads to a 19% reduction of maximum operating frequency with respect to a single access $3R3W$ memory. Nevertheless, as a consequence of the doubled read and write operations per cycle, it is possible to achieve an improving of 62% for memory bandwidth at $0.9\,V$ as supply voltage.
Figure 2.28 shows maximum clock frequency vs total number of RF ports. It is possible to notice that starting from 6 ports, there is a gain in f_{max} by using time multiplexing. This phenomenon can be explained by considering that the effect of the number of ports increasing is destructive on the overall access delay.
Double pumping is very useful to achieve better performance in terms of bandwidth of the memory, but it requires the addition of some modules to properly handle the growth of timing complexity. This means an higher area occupied by the RF on the chip and an higher power consumption, although it is more convenient to implement double pumping rather than doubling bitcell's ports.

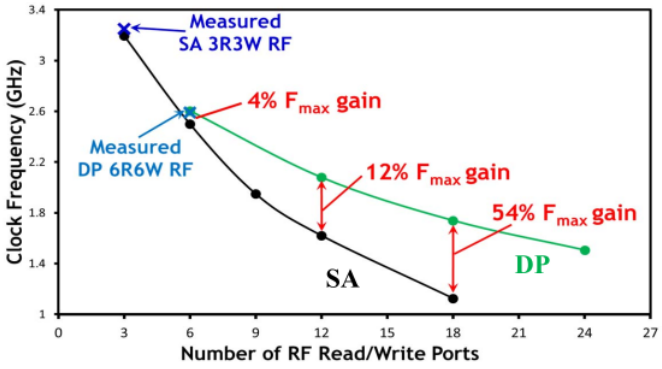


Figure 2.28: Maximum frequency difference between double pumped and single access RFs with different number of ports [3].

2.2.2 Power

Power consumption is a very critical aspect of RFs and in general of ICs design. In a microprocessor, an higher need of data means an higher power dissipation. This could bring to an overheating, so a system reliability drop that must be supplied with some cooling strategies and devices that can negatively affect the cost and the other main features of the system.
Design should be calibrated in order to satisfy some power constraints, both dynamic and static. Dynamic power dissipation can be expressed in a straightforward formulation:

$$P_{dyn} = \alpha * C_L * V_{DD}^2$$

Where α is the switching activity, C_L the load capacitance and V_{DD} the power supply.
For what concerns leakage power, it can be expressed as:

$$P_{leakage} = V_{DD} * I_{leakage}$$

Leakage current depends exponentially on threshold voltage, so a decreasing of threshold provokes a sharp increasing of $I_{leakage}$. With CMOS process node scaling, V_{DD} has been scaled in order to keep power consumption under control, especially dynamic contribution since it depends quadratically on power supply. Simultaneously, it arose the need to scale also V_t in order to

maintain the ratio $\frac{V_{DD}}{V_t}$ as constant as possible to avoid the degradation of performance, but V_t scaling means an increasing of leakage power. In the case of memories, leakage involves bitlines since it reduces their noise immunity [4].

The work reported in [4] proposes a method to reduce bitlines leakage in a $4R4W$ RF by using a $130\,nm$ dual-V_t bulk-CMOS technology. The particular technique is called pseudostatic local bitline, implemented to prevent an excessive drop of the local bitline voltage during evaluation phase. The scheme reported in Figure 2.29 shows how this method has been implemented. It requires the using of 5 additive transistors per read port with respect to a classical scheme in which P_x and NOR gate are not present. When the read select signal RS is not active the presence of the static-precharge transistor P_x fixes the bitline stack node V_s to V_{DD}, so in that case the output of the NOR gate is always 0. Gate-source voltage of $M1$ is equal to $-V_{DD}$, it means a powerful underdrive of each n-channel transistor drain-connected to local bitline during precharge, so subthreshold current is effectively cut off. In addition, the body effect on $M1$ is maximized due to the fact that source-body voltage is equal to power supply, so threshold voltage increases.

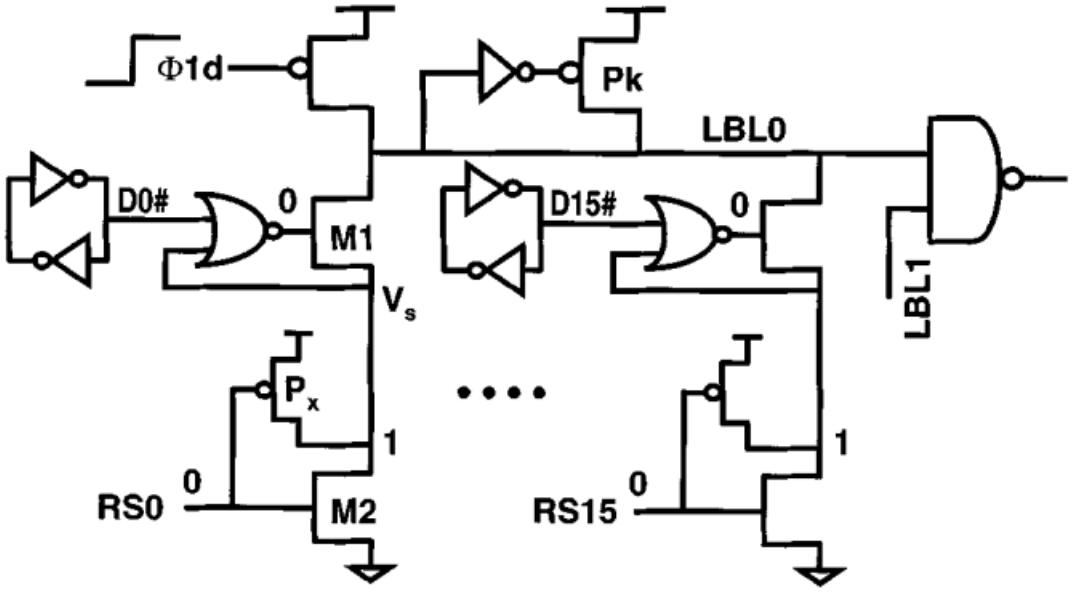


Figure 2.29: Implementation scheme of pseudostatic bitline technique [4].

The application of this technique results in a total active leakage reduction of $703x$, even with using low threshold transistors [4]. Figure 2.30 depicts the behaviour of local bitline during evaluation at $1.2\,V$. The robustness is improved thanks to pseudostatic bitline with respect to a normal architecture without NOR gate and P_x transistor that uses low threshold voltage transistors or two series devices with different thresholds.

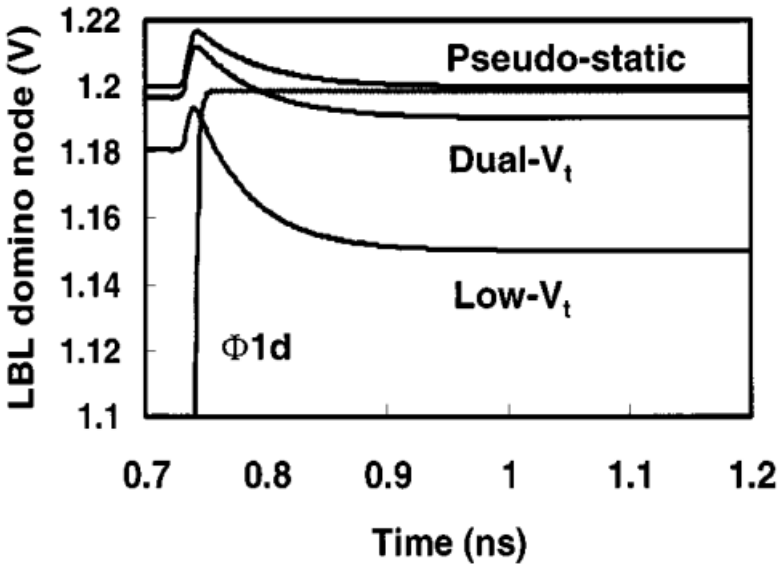


Figure 2.30: Local bitline drop comparison [4].

The reduction of leakage power does not come for free. A 10% area overhead is necessary to integrate static precharge and NOR gate. Moreover, the complete read path operates with a minimum period 4% higher with respect to an optimal-speed solution with low threshold voltage transistors. Finally, the energy/transition, strictly related to dynamic power consumption, including also active leakage, is 7% higher than the optimal-power solution with high threshold voltage transistors due to the increased switched capacitance introduced by additional transistors and their associated active leakage.

2.2.3 Area

Area is a fundamental feature for all ICs since it strongly affects the cost according to the following relation [11]:

$$cost \sim A^4$$

The work proposed in [5] proposes a low leakage bitcell with 4 read ports and 2 write ports that reduces the occupied area. The schematic is shown in Figure 2.31, where only n-channel MOS are used for each read port. This is the reason why the bitcell is marked as $full-N$, used to achieve a lower area on chip than a $1P-3N$ cell. Figure 2.32 shows two examples of $1P-3N$ read port.

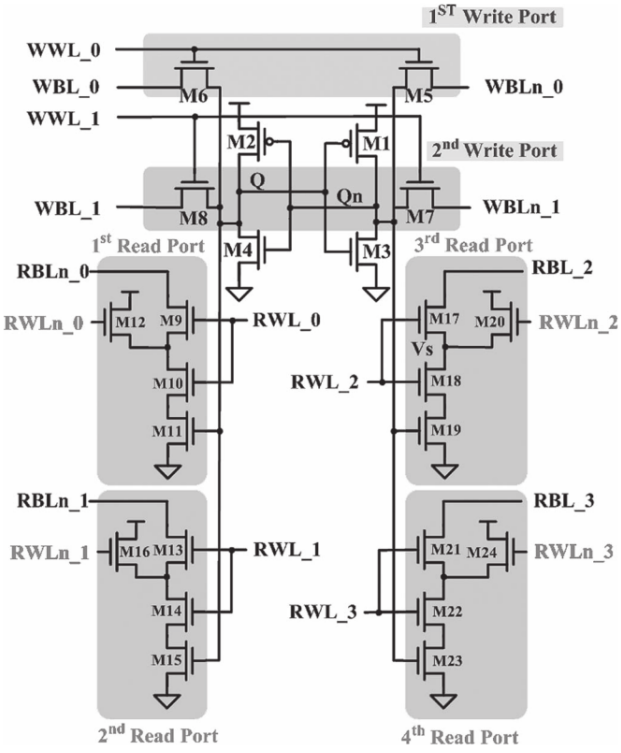


Figure 2.31: Full-N $4R/2W$ bitcell [5].

The leakage of the bitcells connected to a read bitline can destroy its voltage value due to the slow operations in Ultra Low Voltage mode since this RF is designed to be used in a microprocessor that implements the Ultra Dy-

namic Voltage Scaling technique. The proposed solution solves the problem related to leakage by introducing the complementary lines of read wordlines, so RWL_n. While not accessed, leakage is reduced with the enhancement of stacking effect between $M9$ and $M10$ (for first read port) thanks to $M12$ activated by $RWL0\_n$.

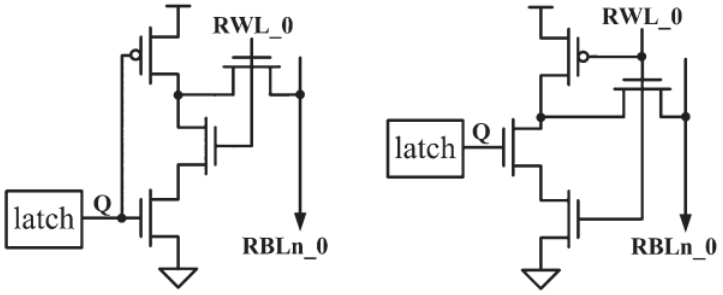


Figure 2.32: $1P - 3N$ read ports [5].

The layout of the two types of cell have been realized by using TSMC 65 nm CMOS process. The sizes are 2.91 x 2.7 μm^2 and 2.88 x 2.23 μm^2 for the full-N. Thus, using full-N bitcell gives an advantage from area point of view. Moreover, this kind of cell provides for a 17 % area overhead with respect to the classical RF cell with 2 nMOS per each read port. Consequently, it is possible to say that a bitcell design focused on the reduction of leakage brings to an increasing of area.

2.2.4 Stability

As explained in Section 2.1.1, the classical configuration of a RF cell is characterized by the fact that hold margin and read margin are equal, with write margin that is the most critical parameter about stability.
A possible solution that improves writability is described in [5], where the *Reconfigurable Write Scheme* is implemented. The principle scheme is depicted in Figure 2.33, which highlights the presence of 2 write ports. When the second port is unused, it is employed to perform the same operation of the first port by writing the same data to the same address.
Simulation has shown a 18 % improvement of write margin thanks to the application of this technique. The extra cost concerns dynamic power and

area. Hence, it implies a $\sim 1.16\,\%$ area overhead and an increasing of $\sim 24\,\%$ of energy/write.

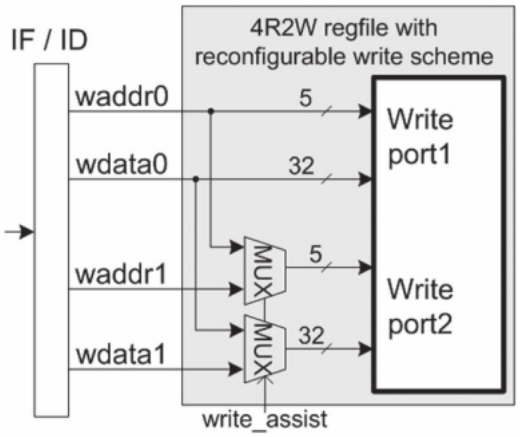


Figure 2.33: Reconfigurable write scheme [5].

Chapter 3

Modules design

The Register File that OpenRAM is able to create after this thesis work follows the basic SRAM structure described in Section 2.1. It consists of the control logic, the bank, the array of D flip flops that receive the address and the array of D flip flops that sample the input data which have to be written inside the bitcells. Control logic contains all is needed to produce the control signals for the rest of the memory, including *clock*, with the correct timing. Bank incorporates bitcell array, row circuitry and column circuitry. The entire structure is supplied with a voltage equal to $1.8\,V$.
There are two possibilities to create the modules that make up all these parts of the entire memory. Some modules, indicated as custom modules or hand-made modules, have been created through an open-source design flow and then integrated in OpenRAM in order to instantiate, place and route them in the correct way. It is necessary to make available the circuit level description (SPICE file) and the graphical level description (GDS file) of the cell that one wants to add inside the produced memory. The second possibility consists in producing a module thanks to OpenRAM framework, which provides the opportunity to create a parametric transistor. In this way, any cell can be made with proper instantiation, placing and connection of the transistors.
In this work, the hand-made modules are:

- Bitcell $2R/1W$
- Bitline sensing circuit

- Write driver
- Row decoder's cell
- Wordline driver's cell
- D - flip flop

All the others have been produced by following the second approach. Actually, OpenRAM had preexisting scripts to create straightforward cells such as inverter, NAND2, NAND3, AND2, AND3, and more complex modules like precharge cell, column multiplexer and control logic. These three last blocks have been modified to be properly interfaced with a multiported bitcell.
The design of custom modules is developed by adopting a top-down approach. It consists of a precise sequence of stages:

1. Definition of input data and output specifications.
2. Paper & pencil calculation.
3. Generation of the schematic of the cell.
4. Evaluation of capacitive load useful for simulation.
5. Simulation of the schematic netlist. If the design meets the output specifications, then it is possible to proceed, otherwise it is necessary to return to step 2.
6. Realization of the layout by using SkyWater130 PDK's design rules.
7. Performing of Layout Versus Schematic.
8. Parasitics extraction.
9. Simulation of the extracted netlist. If the design meets the output specifications, then it is possible to integrate the module to OpenRAM, otherwise it is necessary to return to step 6.

Input data are the features of the transistors provided by Skywater130 PDK and the capacitive load that each module has to drive during a transition. The transistors have been characterized through some simulations reported in Section 3.1, while capacitive loads have been estimated by considering the case of a 32 x 32 Register File realized in Sky130 technology with the configuration planned in this work. The data about transistors' drain and gate capacitances are provided by the last official release of OpenRAM [12]. In particular, the file *tech.py* of Sky130 technology contains the values of minimum gate capacitance and minimum drain capacitance:

$$C_{gate_{min}} = 0.2\,fF$$

$$C_{drain_{min}} = 0.7\,fF$$

Gate capacitance is related to the operating mode of the MOSFET and it is dependent on the physical dimensions of the device. Moreover, its value varies according to the working region. Drain (or diffusion) capacitance is due to the presence of a junction between the diffusion and the substrate. Its calculation involves both an area and a perimeter contributions with a different weight. Diffusion capacitance can be reduced by means of a careful layout. If two transistors of the same type are in series, sharing the diffusion permits to obtain only one capacitive contribution. Moreover, eliminating the contact if it is unnecessary, brings to halve the diffusion capacitance.
For the evaluation of capacitive load, a very approximate model is adopted. It provides for a proportional relation between capacitance and transistor's width. The values reported above refer to a transistor with the minimum width allowed by SkyWater130 PDK design rules, that is $0.42\,\mu m$.
The employed design flow is completely open-source, it involves the following tools:

- Xschem to build the schematics.
- Ngspice to simulate a SPICE netlist.
- Magic to realize the layout.
- Netgen to perform LVS.

Each tool can be freely downloaded from its own Github repository.

3.1 Transistors' characterization

The first step of the design consists in understanding the features of the transistors nMOS and pMOS provided by SkyWater130 PDK. It is a very important stage because it allows to define the input data on which the entire modules design is based, such as the ratio between μ_n and μ_p, electrons and holes mobility respectively, β_n, β_p (after fixing channel width and length) and the threshold voltages. In order to characterize the device, some simulations are performed by setting up a proper measurement scheme shown in Figure 3.1. A DC sweep simulation brings to find the devices' transcharacteristic curves, shown in 3.2 for nMOS and in 3.3 for pMOS. In this case, both devices have a channel width of $1\,\mu m$ and a channel length of $150\,nm$, that is the minimum possible for Sky130 technology. Threshold voltages are found by arbitrarily fixing threshold drain current to $5\mu A$. The values are:

$$V_{tn} = 0.7\,V$$

$$V_{tp} = 0.68\,V$$

Concerning β_n and β_p, they are calculated by considering a set of points in the two curves. Obviously, it is necessary to pay attention to the working region of the transistor. β_n is evaluated in triode and in saturation respectively with the following results:

$$\beta_{n1} = 1\,\frac{mA}{V^2}$$

$$\beta_{n2} = 695\,\frac{\mu A}{V^2}$$

The same procedure is repeated for β_p:

$$\beta_{p1} = 332\,\frac{\mu A}{V^2}$$

$$\beta_{p2} = 252\,\frac{\mu A}{V^2}$$

It is possible to notice that β shows a wide variability range according to the working point of the device. This suggests that the simple analytical commonly used models of the MOSFET are not accurate for these transistors, so it would be better to not rely on those estimations of β.

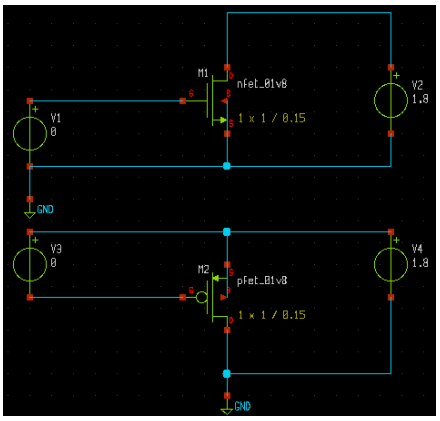


Figure 3.1: Measurement scheme to characterize Skywater130 PDK's nMOS and pMOS.

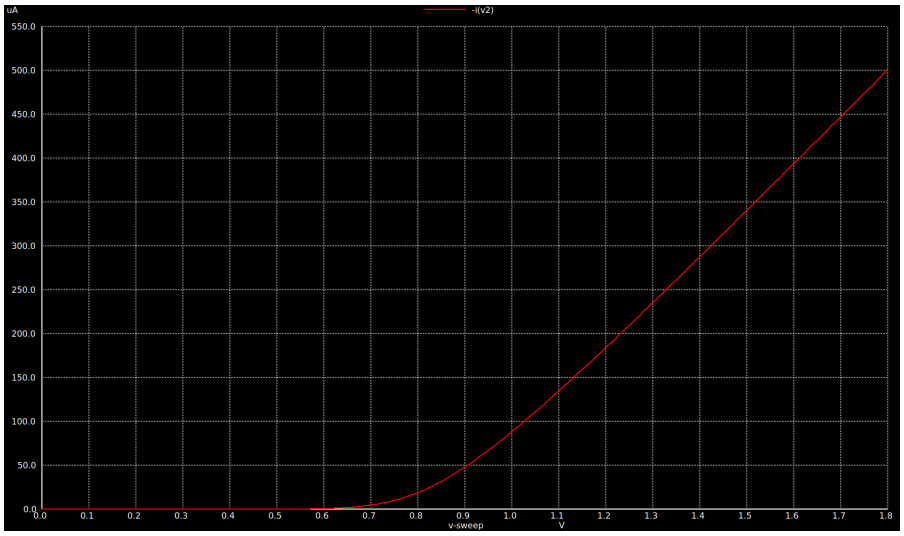


Figure 3.2: Trancharacteristic of a SkyWater130 PDK's nMOS with a channel width of $1\,\mu m$ and a channel length of $150\,nm$.

What is important is the ratio between μ_n and μ_p, useful to size properly the transistors inside the cells. The best way to find this parameter consists

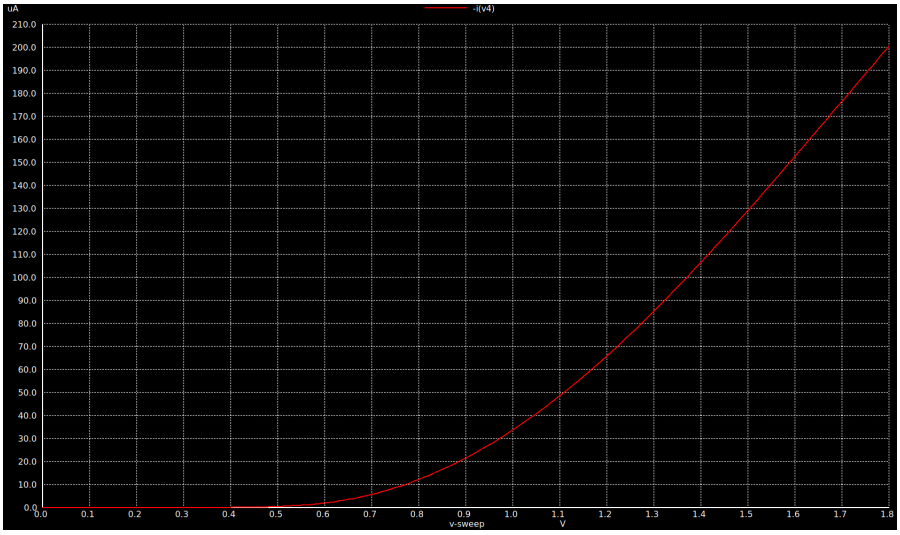


Figure 3.3: Trancharacteristic of a SkyWater130 PDK's pMOS with a channel width of $1\,\mu m$ and a channel length of $150\,nm$.

in testing an inverter by fixing the width of the nMOS to $0.42\,\mu m$ and by changing the width of the pMOS until the input-output curve of the inverter is perfectly symmetrical. This condition is reached for a width of the pMOS equal to $1.35\,\mu m$. Consequently, mobility ratio is equal to:

$$\frac{\mu_n}{\mu_p} = \frac{W_p}{W_n} = 3.2$$

3.2 2R/1W bitcell

The realization of a $2R/1W$ Register File requires the presence of a bitcell with 2 read ports and 1 write port. Its design should be conducted by starting from the input data, that are the known features of the Sky130 PDK's transistors and the output capacitive load to be used in simulation, and the design's aims, represented by enough stability during holding, reading and writing and by the number of ports. Moreover, the layout should ensure an area usage as low as possible.
The requirement about the number of ports can be satisfied by chosen the classical RF cell shown in 2.14, which provides for a single ended reading and a differential writing. As explained in Section 2.1.1, this cell ensures a read

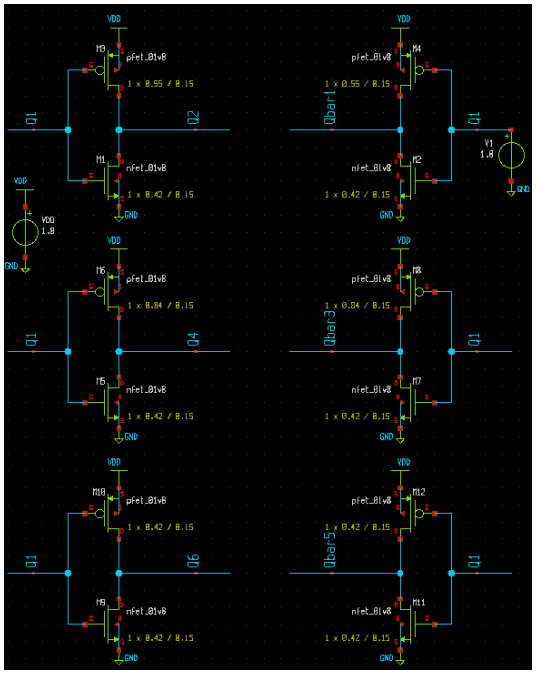


Figure 3.4: Simulation scheme to evaluate hold margin.

margin equal to hold margin, while write margin is the most critical aspect about stability.

First of all, it is neccesary to focus on the design of the bitcell. The first step provides for a correct sizing of the transistors of the cell by performing some tests concerning stability evaluation. The first simulation aims to evaluate hold margin and it is executed by using the scheme depicted in Figure 3.4, which shows three different possibilities to size the transistors of the coupled inverters. By DC sweeping $V1$ from $0\,V$ to $1.8\,V$, it is possible to find the results in Figure 3.5. The best condition is the one for which:

$$W_p = 0.84\,\mu m$$

$$W_n = 0.42\,\mu m$$

Regarding write stability, the scheme involved in the test is the one in Figure

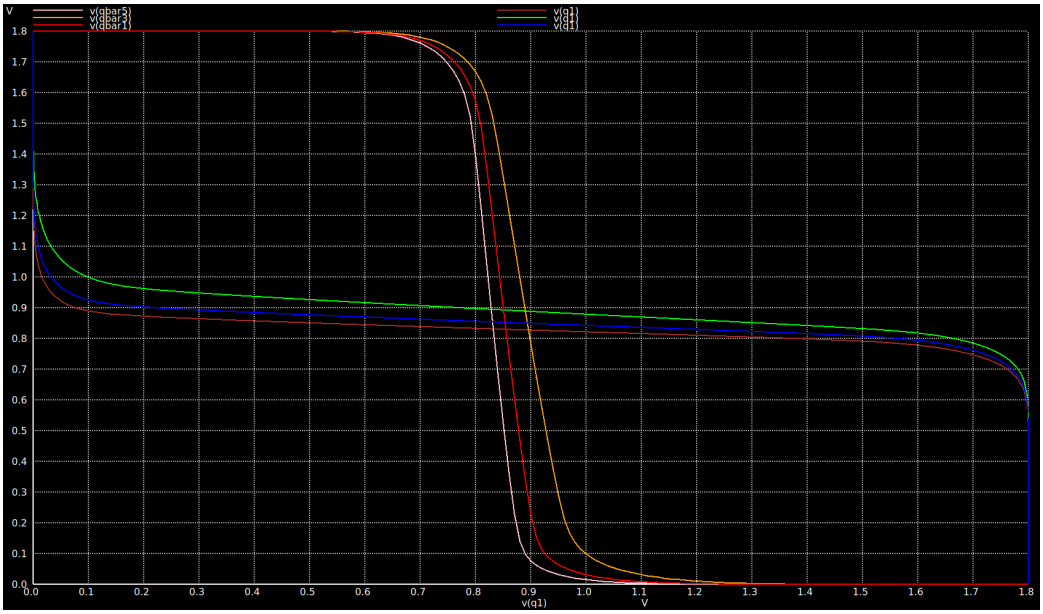


Figure 3.5: Results of the simulation to evaluate hold margin.

3.6. The best condition is realized when:

$$W_p = 0.84\,\mu m$$

$$W_n = 0.42\,\mu m$$

$$W_{writeaccess} = 0.42\,\mu m$$

The inverter used produce the complementary of write bitline is sized in such a way the width of the pMOS is equal to $0.84\mu m$ and the width of the nMOS is equal to $0.42\mu m$. It is a good compromise between area and balance of the inverter itself.
Finally, the read access transistors are sized. Since read margin is not a problem in this kind of bitcell, sizing is related only to the time needed to discharge the read bitline, so it is a matter of performance. This is the reason why the transistors in each read port have a width doubled with respect to the minimum one. In this way, the time needed to read a bit is reduced by 55% with the same bitline capacitance.
The schematic of the entire bitcell can be found in Figure 3.8.
The output capacitance estimation is based on the case of a 32 x 32 RF,

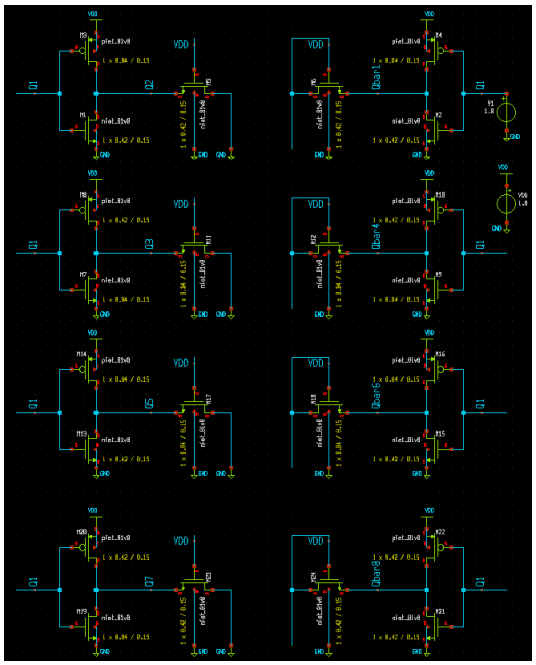


Figure 3.6: Simulation scheme to evaluate write margin.

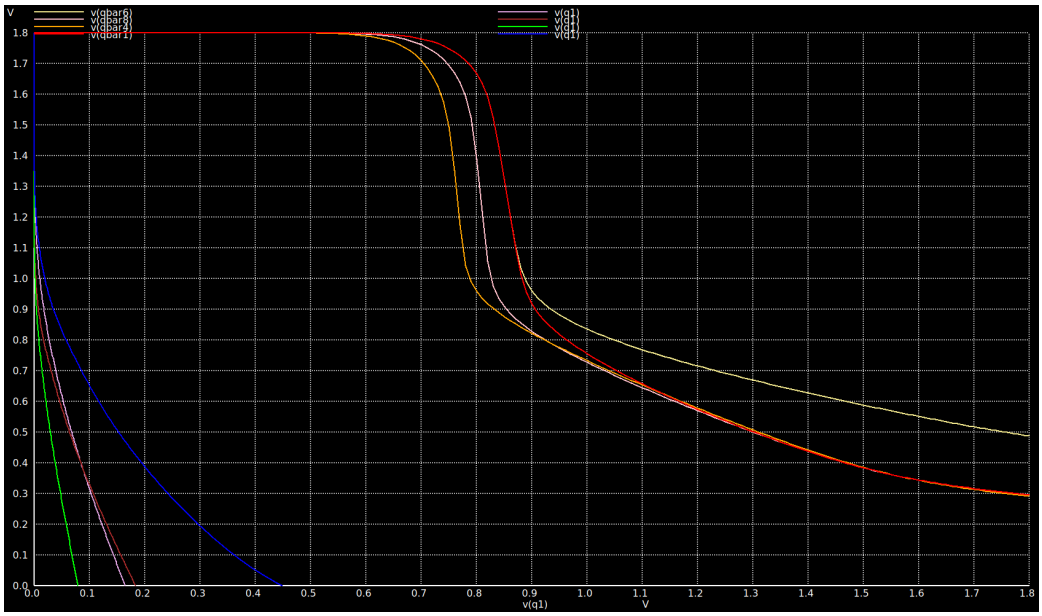


Figure 3.7: Results of the simulation to evaluate write margin.

whose bitcell array is organized in 32 rows of 32 bits. It means that it is enough to evaluate the load of a single read/write port and then multiplying it by 32 to obtain the total load on the bitline. With regard to read bitline, a read port is composed by two series transistors with a width doubled with respect to the minimum one. Therefore, diffusion capacitance of each read port's transistor is equal to:

$$C_d = \frac{0.84\,\mu m}{0.42\,\mu m}\,0.7\,fF = 1.4\,fF$$

At this point, it is necessary to consider the contributions of the other bitcells on the same column that share the same read bitline. The capacitive load is the one of the drain of the transistor connected to read bitline. Another contribution to be considered is provided by the module driven by read bitline. In a 32 x 32 RF there is only 1 word per row, so column multiplexing is not necessary. It means that read bitline is connected to the input of a sense amplifier, that, as explained in Section 3.3.1, is an HI skewed inverter. Therefore, the total capacitive load on a read bitline evaluated for a 32 x 32 RF is equal to:

$$C_{total\_read\_bitline} = 2\,C_d + \frac{2.7\,\mu m}{0.42\,\mu m}\,C_{gate_{min}} + C_{gate_{min}} = 46.3\,fF$$

Write bitline capacitive load can be estimated in the same way of read bitline. For a single activated write bitline, it is necessary to consider the contributions of the inverter that makes writing differential, the pass transistor and the write driver. The write driver, as explained in Section 3.3.2, has a balanced tristate inverter with double strength as output stage. So, capacitive load of a write bitline can be evaluated as:

$$C_{single\_write\_bitline} = 3\,C_{gate_{min}} + C_{drain_{min}}$$

$$+8.4\,C_{drain_{min}} = 4.7\,fF$$

In order to evaluate the total load of the write bitline, the contribution of the bitcell that are not activated is needed. They provide the input capacitance

of the inverter and the diffusion capacitance of a pass transistor.

$$C_{total\_write\_bitline} = C_{single\_write\_bitline} + 31\,(3\,C_{gate_{min}}$$

$$+C_{drain_{min}}) = 45\,fF$$

The loads used for the bitcell simulation don't involve the contributions of the bitcell itself, but they have to contain only the external capacitive contributions. It means that the used values are:

$$C_{rbl\_sim} = C_{total\_read\_bitline} - C_d = 44.9\,fF$$

$$C_{wbl\_sim} = C_{total\_write\_bitline} - (3\,C_{gate_{min}} + C_{drain_{min}}) = 43.7\,fF$$

At this point, the bitcell is tested through a transient analysis by performing a writing of a 0, then a writing of a 1 and a subsequent reading. The waveforms are depicted in Figure 3.9. All the input signals are not ideal, they have both rise and fall time equal to $50\,ps$. While reading a 1, read bitline is discharged with a speed that depends strongly on bitline capacitance. It means that the parasitic capacitance on the read bitline deeply affects the performance of the entire memory. The measured delays are the read bitline's fall time from 90 % to 10 % and the delay between the write bitline signal's and the stored data's transitions at 50 %. They are respectively equal to:

$$t_{fall\_rbl} = 337\,ps$$

$$t_{wbl\_to\_Q} = 70\,ps$$

After testing the bitcell and verifying that it fits all the specifications, it is possible to realize the layout by using Magic. This powerful tool is very convenient because it performs a real time Design Rule Check (DRC).
There are few constraints about the realization of the layout. The minimization of the area is one of them, the other requirements originate by considering the connections between the bitcells inside the array and between the array and row and column circuitry.
First of all, the layers for bitlines and wordlines must be chosen once for all

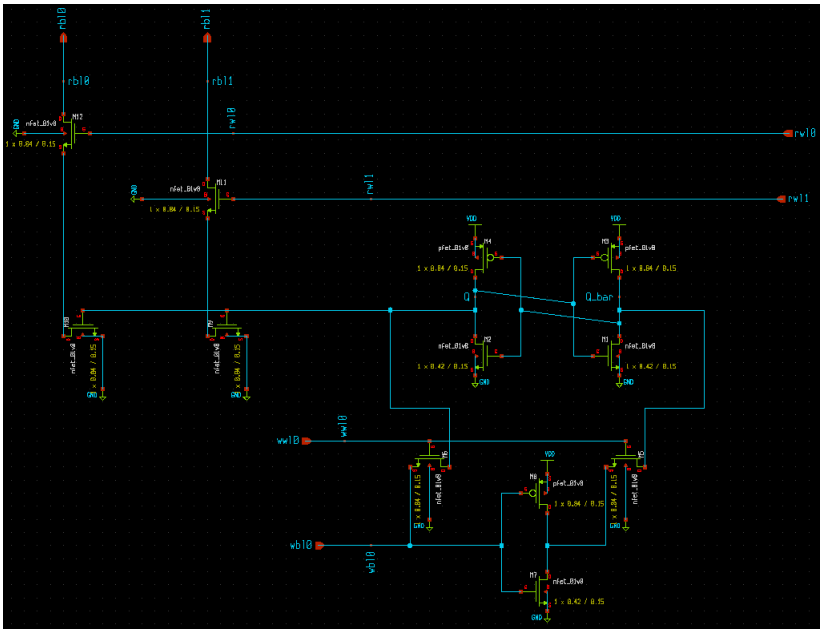


Figure 3.8: Schematic of the bitcell used in the design of the $2R/1W$ Register File.

since they are connected to other modules and there must be layers compatibility. In this case, bitlines are realized in metal 2, while wordlines in metal 1. It means that row circuitry has metal 1 outputs, while all the blocks that compose column circuitry have inputs and/or outputs in metal 2. In the bitcell layout it is necessary to pay attention to the fact that no metal 1 must be present to the right and to the left of a wordline contact along all the width of the bitcell, otherwise the connection between the wordline contacts of the bitcells inside a row and the corresponding outputs of the row circuitry would cause either some short circuits that destroy all the design or some design rules violations due to not enough spacing between two metal1 lines. The resulting area is $8.29\,\mu m$ x $3.89\,\mu m = 32.25\,\mu m^2$. Then, Magic terminal console is used to get the extracted netlist, which is compared with the netlist obtained from the schematic to perform LVS through Netgen. Since the two netlists match uniquely, then extracted netlist is obtained again by setting the Magic console's *ext2spice* command's options *cthresh* and *rthresh* equal to 0. In this way, all the parasitic capacitances and resistances are annotated on extracted netlist in order to perform simulation. The measured delays

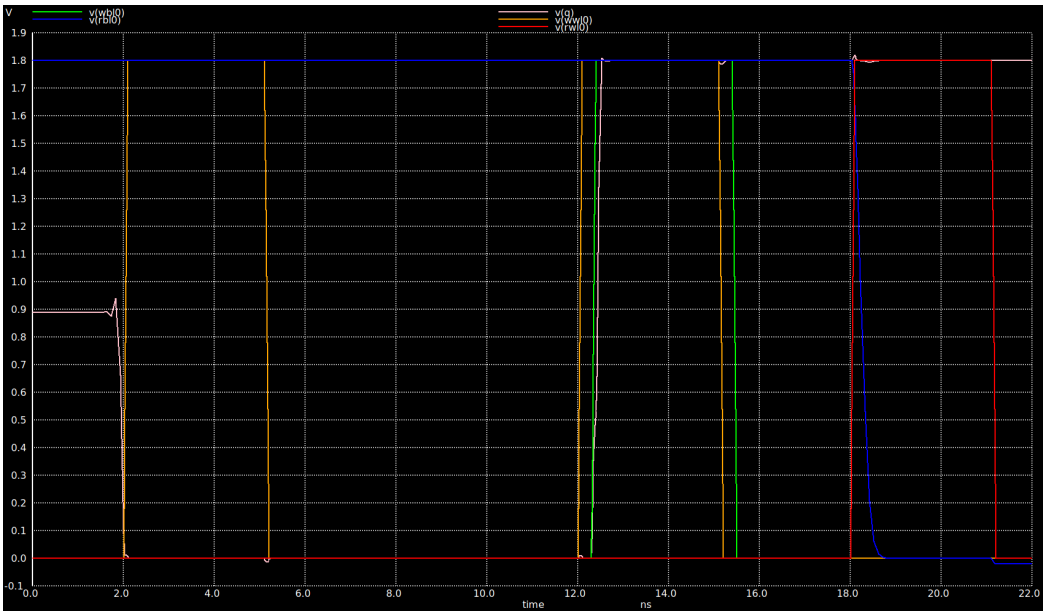


Figure 3.9: Waveforms representing write and read operations of the $2R/1W$ bitcell.

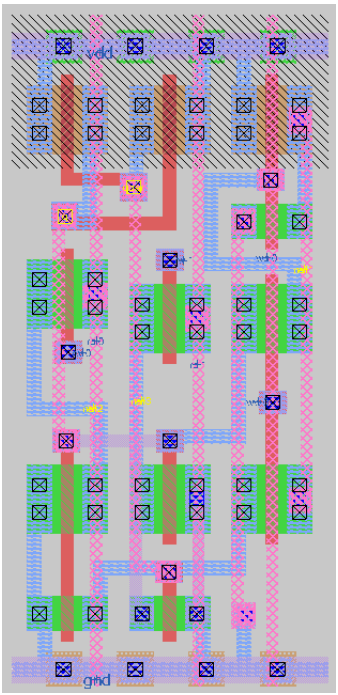


Figure 3.10: Layout of the $2R/1W$ bitcell.

are:

$$t_{fall\_rbl} = 408\,ps$$

$$t_{wbl\_to\_Q} = 139\,ps$$

The extracted netlist is also used to evaluate the features of the cell from power point of view. Medium power dissipation is estimated thanks to a SPICE simulation during reading from one port and writing of a 0 and of a 1. Moreover leakage power is reported as well. The obtained results are:

$$P_{read\_0} = 0.18\,\mu W$$

$$P_{write\_0} = 13.8\,\mu W$$

$$P_{read\_1} = 2.95\,\mu W$$

$$P_{write\_1} = 23.9\,\mu W$$

$$P_{leakage} = 0.32\,\mu W$$

Reading a 1 is much more power consuming because the read bitline is discharged through the path offered by the transistor gate connected to the stored data.

3.3 Column circuitry

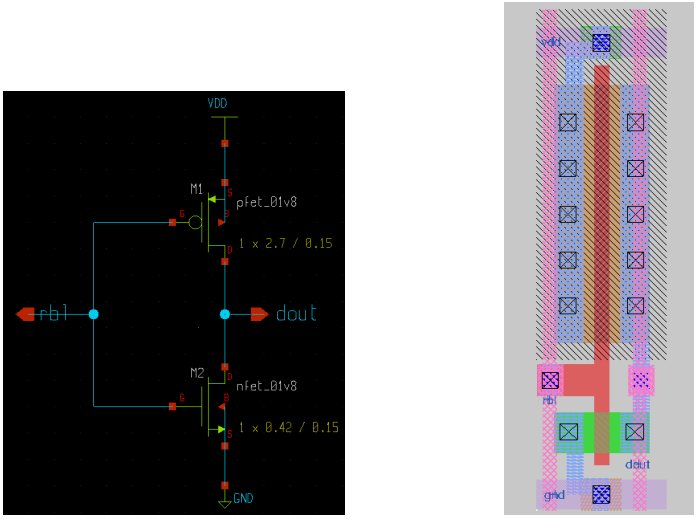
3.3.1 Bitline sensing circuit

The chosen bitcell provides for a single ended read procedure, so, in order to not complicate the design and to save area, the adopted sensing strategy is single ended. In particular, the circuit is composed by a simple HI skewed inverter which receives the read bitline and produces the output data. Obviously, the bitline value is inverted but it is coherent with the used bitcell since the value that it is possible to find on the read bitline is inverted with respect to the one stored by the bitcell itself.
Next step consists in focusing on the sizing of the inverter by considering the ratio between electrons and holes mobility roughly equal to 3.2 and the fact that the inverter has to be realized in order to speed up a low to high

transition by keeping an eye on area.
The proposed scheme is the one in Figure 3.11a characterized by a pull down transistor having the minimum width and a pull up transistor with a width of $2.7\mu m$.
Then, the input capacitive load useful for simulation is evaluated. The input data of the bitline sensing circuit could come from the column mux or the bitcell, but in a 32 x 32 RF, column multiplexing is not present, so only the second possibility will be considered. In order to evaluate input load, it is necessary to take into account the total capacitance on a read bitline reported in Section 3.2 and then subtracting the input capacitance of the bitline sensing circuit. In this way, the input load used in the simulation is:

$$C_{in\_bit\_sense} = C_{total\_read\_bitline} - (\frac{2.7\,\mu m}{0.42\,\mu m} C_{gate_{min}} + C_{gate_{min}}) = 44.8\,fF$$

Concerning output capacitance, the output node of the bitline sensing circuit is one of the output pins of the memory. It means that the output capacitance depends on what block receives the output data. For the simulation, a load value is chosen by considering 10 times the minimum gate capacitance of Sky130 transistor. Two kinds of simulation are performed to ensure that the



(a) Schematic of the bitline sensing circuit.

(b) Layout of bitline sensing circuit.

Figure 3.11: Schematic and layout of bitline sensing circuit.

circuit meets the specifications. The first one is a DC analysis, whose result is shown in Figure 3.12, that highlights the inverter's trend to favour a low to high transition. The second simulation, Figure 3.13 is based on a transient analysis that allows to measure four delay times:

$$t_{LH_{50}} = 36\,ps$$

$$t_{HL_{50}} = 78\,ps$$

$$t_{rise_{10-90}} = 54\,ps$$

$$t_{fall_{10-90}} = 117\,ps$$

These values are dependent on the rise and fall times of the input signal. In this case, the input signal is not ideal, it has rise and fall times equal to $50\,ps$.

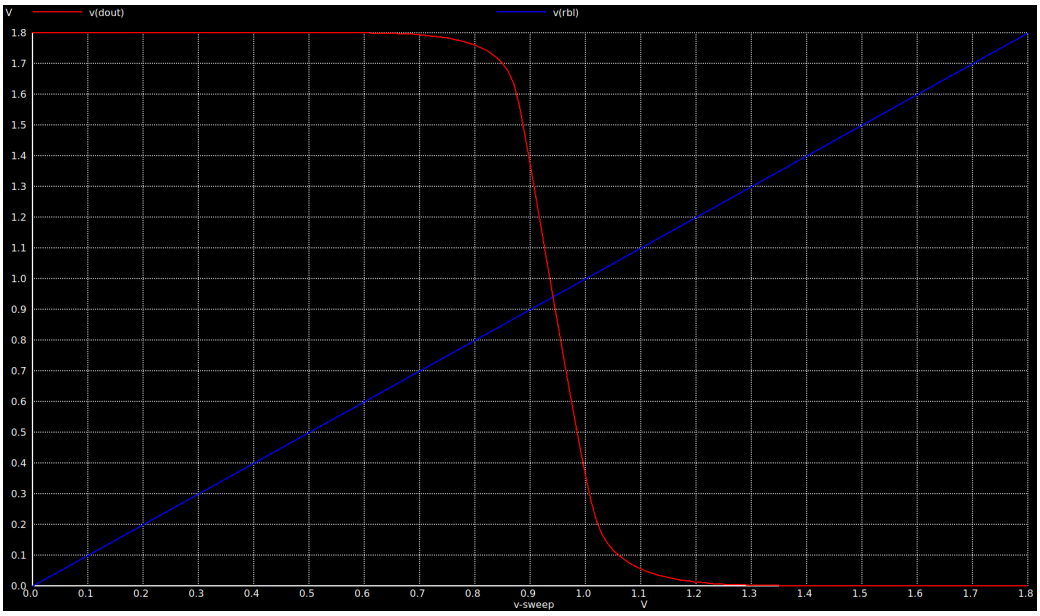


Figure 3.12: Results of the DC simulation of the bitline sensing circuit.

Since the presented schematic satisfies the output specifications, then it is possible to produce the layout. There are only two constraints about the layout of the bitline sensing circuit: input and output lines must be realized in metal 2 and the width of the cell must be lower than an half of the bitcell's

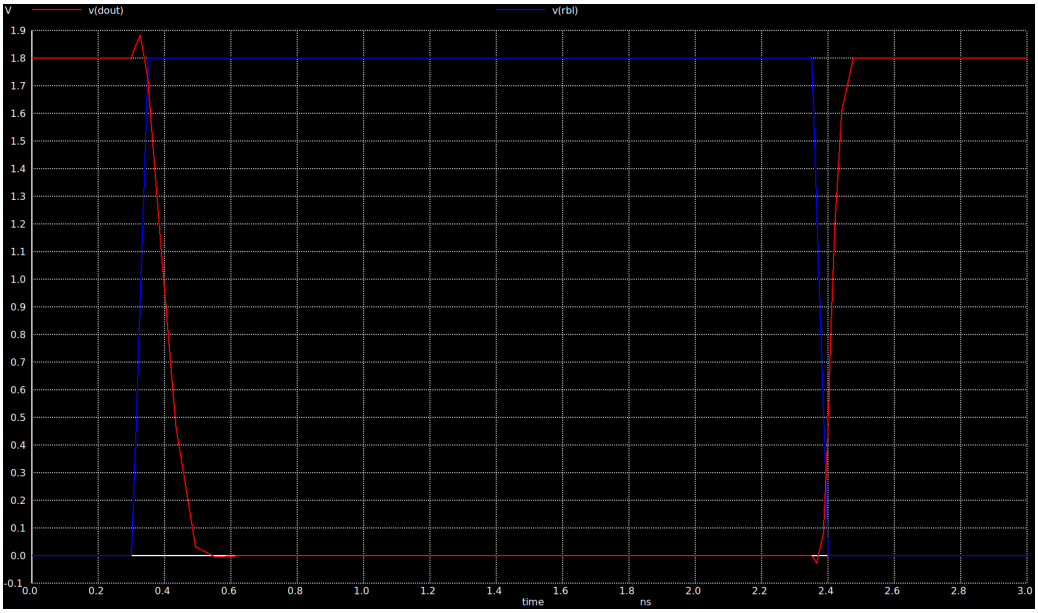


Figure 3.13: Results of the transient simulation of the bitline sensing circuit.

width plus an half of the minimum spacing between two nwells (equal to $1.27\,\mu m$). The former is due to the fact that bitline sensing circuit's receives either the read bitline, metal 2 line, or the output of the column multiplexer, realized in metal 2 as well. The latter is related to the fact that each bitcell has two read bitlines, so it is necessary to place two bitline sensing circuits per bitcell (or per each bit per word in case of column multiplexing). In order to avoid that the array containing the bitline sensing circuits placed horizontally is larger than the bitcell array, it is necessary that the sum between the total width of two side to side bitline sensing circuits and the minimum spacing between two nwells dictated by design rules is lower than the bitcell's width.
The realized layout is shown in Figure 3.11b. It has an area of $5.25\,\mu m$ x $1.05\,\mu m = 5.51\,\mu m^2$. After a successful LVS, the extracted netlist is simulated in the same condition of the previous simulation. The resulting four delays are:

$$t_{LH_{50}} = 38\,ps$$

$$t_{HL_{50}} = 81\,ps$$

$$t_{rise_{10-90}} = 56\,ps$$

$$t_{fall_{10-90}} = 118\,ps$$

Once again, extracted netlist is used to obtain results about power. The aim consists in evaluating the power associated to an output high to low transition followed by a low to high transition since it is the working method of the bitline sensing circuit when reading a 1. When the input data is a 0, sensing circuit doesn't commute because the bitline is already precharged. The results are:

$$P_{tran} = 69.35\,\mu W$$

$$P_{leakage} = 60\,pW$$

3.3.2 Write driver

Write driver is a module needed to perform a correct writing of a bitcell. It receives a signal coming from control logic that enables the write operation, and a signal coming from the data D flip flop that identifies the input data. This function can be implemented by a tristate inverting buffer, but there are two problems to face. The first one regards the need to create the complementary of the enable signal since control logic gives only the enable signal itself, the second one concerns the fact that it would be an inverting stage, so the voltage level on the write bitline would be the opposite with respect to the one of the signal provided by the external. Those issues can be easily solved through the insertion of two inverters, one to invert the enable signal and the other one to prevent a wrong voltage value on write bitline.
The schematic is shown in Figure 3.15, where the transistors are sized in order to obtain a balanced response of the entire cell, so both transitions should have the same speed. The key point is the non minimum dimension of the final stage's transistors. The write driver is implemented with a strength four times higher with respect to the minimum one in order to solve the conflict arising during writing in all the process corners.
Regarding load evaluation, input load on data pin is estimated by considering the D flip flop structure described in Section 3.5, while output load is the one of a write bitline reported in Section 3.2 minus the write driver contribution.

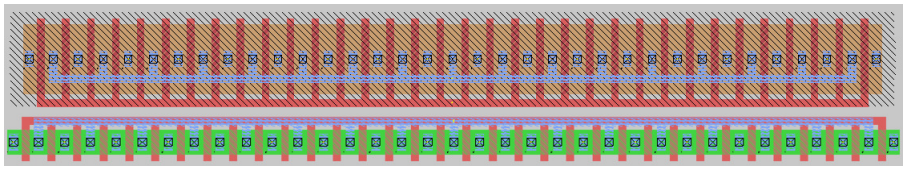


Figure 3.14: Transistors of the output stage of the control logic's block that provides enable to write drivers.

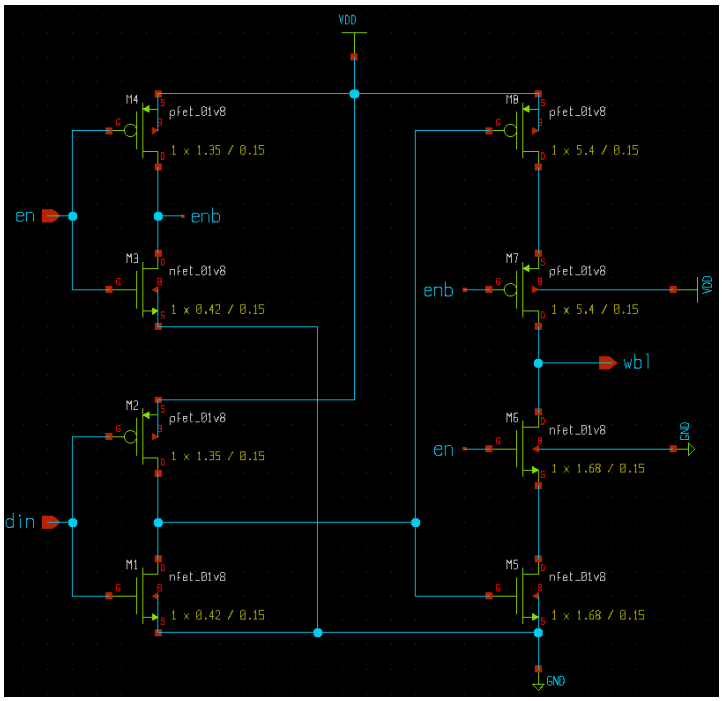


Figure 3.15: Schematic of the write driver.

Enable pin is provided by control logic and in particular by an AND gate whose nMOS width is $17\,\mu m$, while pMOS width is $50.32\mu m$. The devices are shown in Figure 3.14

$$C_{din\_write\_driver} = 4.2C_{drain_{min}} = 2.94\,fF$$

$$C_{en\_write\_driver} = 80C_{drain_{min}} = 56\,fF$$

$$C_{out\_write\_driver} = 32(3C_{gate_{min}} + C_{drain_{min}}) = 41.6\,fF$$

The write driver is tested in such a way write bitline firstly takes an high

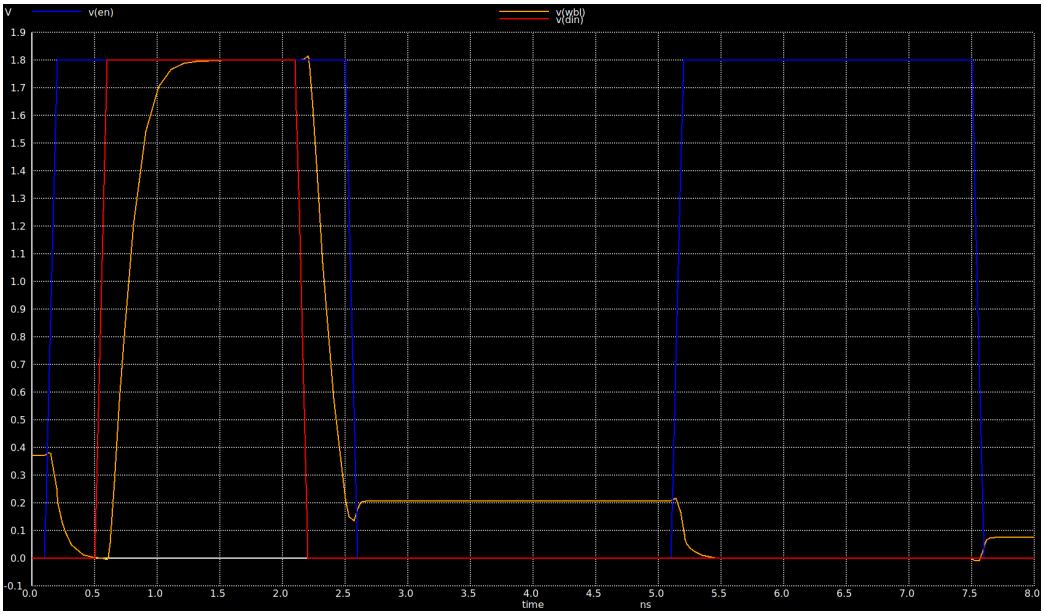


Figure 3.16: Simulation of the write driver.

voltage level followed by an high to low transistion. The results of that simulation are reported in Figure 3.16 and it allows to measure $t_{rise_{10-90}}$ and $t_{HH_{50}}$. Then, another simulation is performed to measure also the other two delays. All the tests are done by considering non ideal input signals with rise and fall times of $50\,ps$. The measured delays are:

$$t_{HH_{50}} = 177\,ps$$

$$t_{LL_{50}} = 163\,ps$$

$$t_{rise_{10-90}} = 188\,ps$$

$$t_{fall_{10-90}} = 157\,ps$$

The layout is realized by considering the following constraints: the sum between the width of the cell and the minimum spacing between nwells must be smaller than the width of the cell for the same reason explained in Section 3.3.1; the output of the write driver must be realized in metal 2 since both column multiplexer inputs and write bitline are metal 2 lines; the en-

able signal of the write driver must have a metal 1 interfacing since metal 1 is used to drive that signal from control logic; input data must have a metal 2 interfacing, the same layer of the output line of the data D flip flop.
Unfortunately, it was not possible to satisfy the requirement about the maximum width. Then, in order to avoid the violation of the minimum distance between nwells in the array containing write driver, the cell is designed with a width equal to the one of the bitcell. In this way, write driver array and bitcell array have the same width and all the write drivers share the nweel.
The layout is depicted in Figure 3.17, it satisfies all the constraints. The area is equal to 10.07 μm x 3.89 μm = 32.17 μm^2. Once obtained a successful

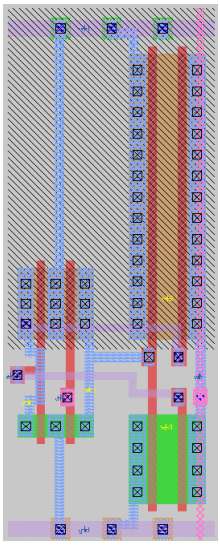


Figure 3.17: Layout of the write driver.

LVS, the extracted netlist is tested in the same way as the the schematic netlist and delays are measured:

$$t_{HH_{50}} = 192\,ps$$

$$t_{LL_{50}} = 174\,ps$$

$$t_{rise_{10-90}} = 210\,ps$$

$$t_{fall_{10-90}} = 204\,ps$$

The output of the write driver can assume either low or high voltage value, so power dissipation during the transition is evaluated in both cases. Moreover, leakage power is estimated.

$$P_{wbl=0} = 79\,\mu W$$

$$P_{wbl=1} = 79\,\mu W$$

$$P_{leakage} = 26\,nW$$

3.4 Row circuitry

In the native version of OpenRAM, row circuitry is composed by a row decoder and wordline drivers. Row decoder is realized through predecoding technique, with predecoders' outputs connected to AND gates with 2 or 3 inputs. All the cells in row circuitry are generated by OpenRAM's framework and then a certain number of them is instantiated. AND gates have the same height of the preexisting single port cell. If this strategy is kept for the realization of the $2R/1W$ Register File, the row decoder will have a total height much higher than the one of the bitcell array because each bitcell has 3 wordline's inputs, so in the row decoder there would be a number of AND gates (one above the other) 3 times higher than the number of the bitcells in a column. Same thing concerning wordline driver array. This condition would be not compatible with the rule for which row decoder and bitcell array have to be pitch matched.
In order to satisfy that constraint, a different strategy is adopted. The predecoders are kept as in the native version of the memory compiler, while row decoder's AND gates and wordline driver are hand-made. Predecoders follow the same topology shown in Figure 2.16.

3.4.1 Row decoder cells

3 inputs AND gates

AND gate with three inputs is realized by adopting static CMOS logic. It consists of a three inputs NAND gate whose output is connected to the

input of an inverter. The schematic is depicted in Figure 3.18. The design of transistors' size is based on the idea to obtain a balanced gate, so theoretically nMOS width should be $0.42\,\mu m$ while pMOS width should be $\frac{1}{3}\frac{\dot{\mu_n}}{\mu_p}\dot{0}.42\mu m = 0.45\,\mu m$. Actually, simulation gives results not coherent with paper & pencil calculations since it shows the behaviour of a HI skewed gate. A proper sizing is the one shown in the schematic, with $W_n = 1\,\mu m$ and $W_p = 0.42\,\mu m$.
The discrepancy between paper & pencil design and simulation results highlights the inaccuracy of the inversely proportional relation between MOSFET's channel resistance and channel width. Moreover, series MOSFETs cannot be considered as series resistances.
Once again, input and output load evaluation is performed by considering the blocks connected to this gate. The 3 inputs of the AND gate are the outputs of the predecoder, whose output stage is a balanced inverter. Regarding output load, the output signal enters as an input on wordline driver, that, as described in 3.4.2, is a 2 inputs AND gate. It means that the output capacitance is the input load of a AND 2 with a pull-down size of $0.55\,\mu m$ and a pull-up size of $0.42\,\mu m$.

$$C_{in\_AND3} = 4.2\,C_{drain_{min}} = 2.94\,fF$$

$$C_{out\_AND3} = 2.3\,C_{gate_{min}} = 0.46 fF$$

By considering three non ideal input signal with rise and fall times equal to $50\,ps$ and that commute at the same time, the characterizing delays of the AND 3 are:

$$t_{HH_{50}} = 59\,ps$$

$$t_{LL_{50}} = 59\,ps$$

$$t_{rise_{10-90}} = 30\,ps$$

$$t_{fall_{10-90}} = 28\,ps$$

Since the output of the AND gate is a wordline (by neglecting the presence of the wordline drivers for the moment), row decoder should have three AND

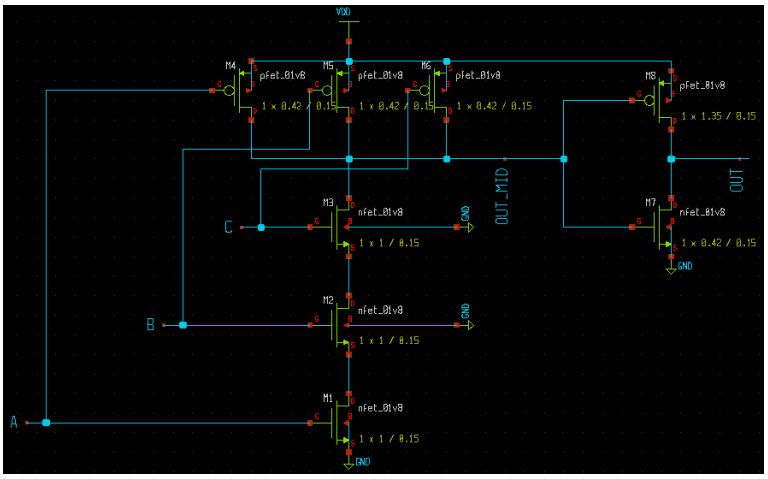


Figure 3.18: Schematic of the 3 inputs AND gate.

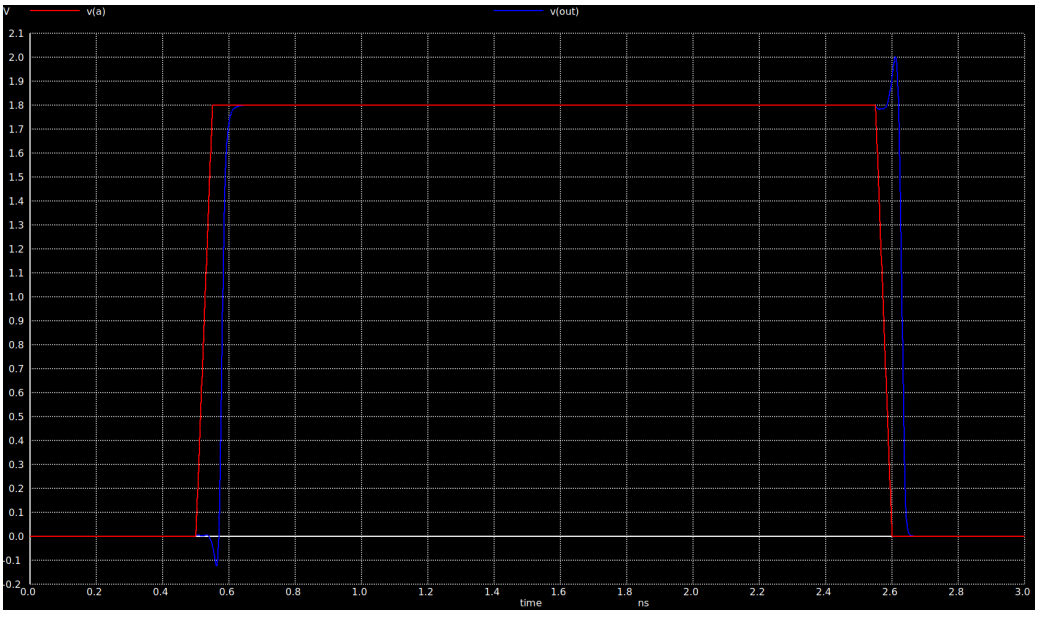


Figure 3.19: Simulation of the 3 inputs AND gate.

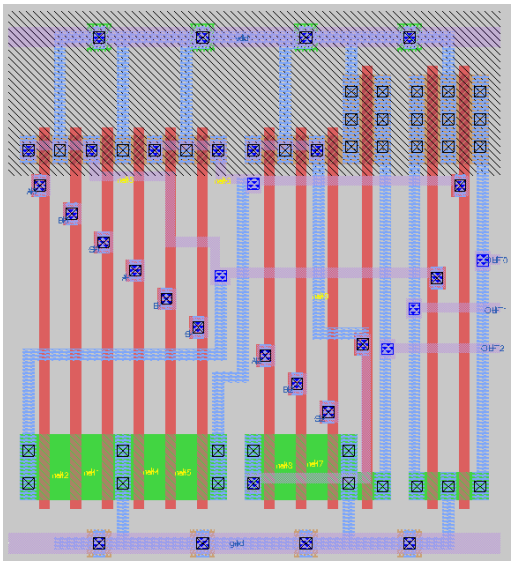


Figure 3.20: Layout of the cell used in row decoder with three AND3 gates.

gates per each bitcell one above the other. It means that, in order to have a pitch matched row decoder, the height of the AND gate must be the one of the bitcell divided by 3. It would be not possible to obtain such a result. The adopted solution consists in merging three AND3 gates in only one row decoder cell. All the cell's transistors are placed in the horizontal direction, so maximum height constraint is not a problem anymore. Each cell has 9 inputs and 3 outputs, one per each bitcell's wordline. Another layout requirement is related to the layer used for input and output pins. Input pins must be realized in metal 1 since the outputs of the predecoders are metal 1 lines. Output pins' layer can be freely chosen since they have to be connected as inputs of wordline drivers, that are custom modules. The adopted layer is metal 1.
The layout is shown in Figure 3.20. Area is equal to $8.29\,\mu m$ x $7.47\,\mu m = 62\,\mu m^2$. Once again, after LVS the extracted netlist is and it gives coherent results with respect to the previous simulation. The measured delays are:

$$t_{HH_{50}} = 88\,ps$$

$$t_{LL_{50}} = 90\,ps$$

$$t_{rise_{10-90}} = 40\,ps$$

$$t_{fall_{10-90}} = 37\,ps$$

Power dissipation across the transitions depicted in Figure 3.19 and leakage power are measured.

$$P_{tran} = 448\,\mu W$$

$$P_{leakage} = 0.58\,\mu W$$

2 inputs AND gates

The design of the AND gate with 2 inputs is conducted in the same way as the one of the AND3 gate. It is composed by a 2 inputs NAND gate followed by an inverter. Theoretical sizing brings to design a nMOS channel width of $0.42\,\mu m$ and a pMOS channel width of $3.2 \cdot 0.42/2\,\mu m = 0.672\,\mu m$. Once again, simulation reported some results that differ with respect to the expected ones. Hence, the resulting gate behaves like a HI skewed AND gate, that is not coherent with the target. By changing transistors' size, things get better. In particular, pMOS channel width is fixed to $0.42\,\mu m$, while nMOS width is set to $0.55\,\mu m$.
Input and output loads are the same of AND 3 gate since the inputs are provided by predecoder and the output is an input signal of wordline driver.

$$C_{in\_AND2} = 4.2\,C_{drain_{min}} = 2.94\,fF$$

$$C_{out\_AND2} = 2.3\,C_{gate_{min}} = 0.46fF$$

Simulation results are shown in Figure 3.22. By considering two non ideal input signals with rise and fall times equal to $50\,ps$ that commute at the same time, the measured delays of the AND2 are:

$$t_{HH_{50}} = 58\,ps$$

$$t_{LL_{50}} = 60\,ps$$

$$t_{rise_{10-90}} = 36\,ps$$

$$t_{fall_{10-90}} = 32\,ps$$

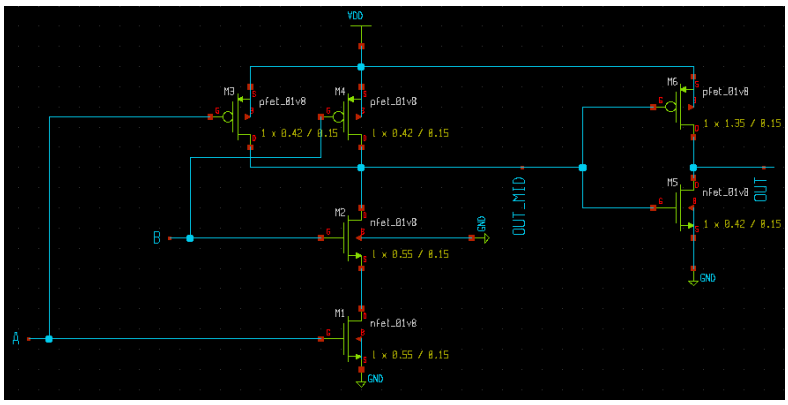


Figure 3.21: Schematic of the 2 inputs AND gate.

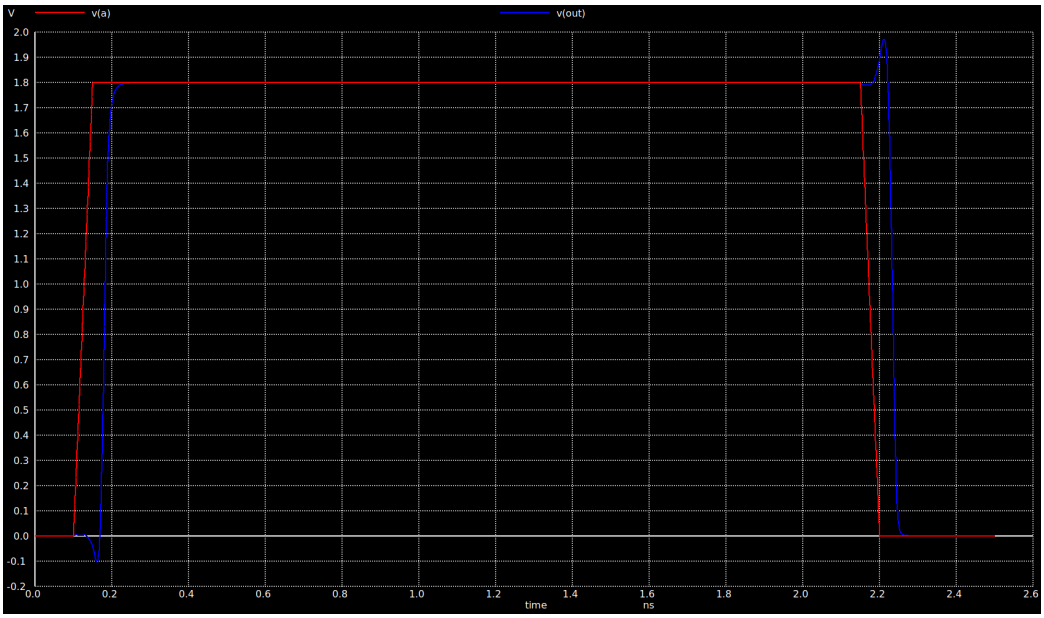


Figure 3.22: Simulation of the 2 inputs AND gate.

The layout of row decoder's cell containing AND 2 gates is realized in the same way with respect to the one of the previous cell. It is shown in Figure 3.23. Area is equal to 8.29 μm x 6.02 μm = 49.9 μm^2. After a successful LVS,

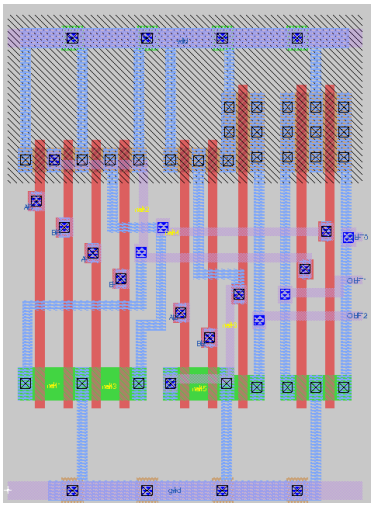


Figure 3.23: Layout of the cell used in row decoder with three AND2 gates.

the cell is tested again and the resulting delays for AND 2 gate are:

$$t_{HH_{50}} = 95\,ps$$

$$t_{LL_{50}} = 98\,ps$$

$$t_{rise_{10-90}} = 47\,ps$$

$$t_{fall_{10-90}} = 47\,ps$$

Once again, power dissipation across the transitions depicted in Figure 3.22 and leakage power are measured.

$$P_{tran} = 65\,\mu W$$

$$P_{leakage} = 0.45\,\mu W$$

3.4.2 Wordline driver

Wordline driver is the second module of the row circuitry. It receives the output of the row decoder and, if enabled, provides the wordline to bitcell array. The chosen topology to satisfy these specifications consists of a balanced 2 inputs AND gate like the one designed for row decoder in Section

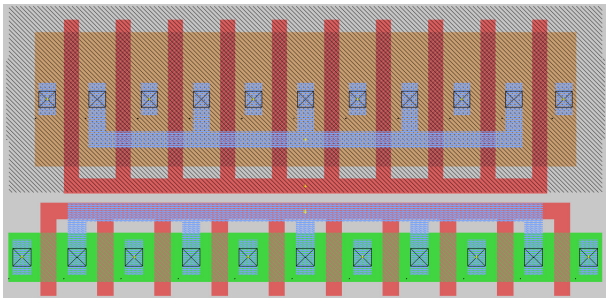


Figure 3.24: Transistors of the output stage of the control logic's block that provides enable to wordline drivers.

3.4.1.

Input and output load is the only difference between the row decoder's AND 2 gate and the wordline driver. Wordline driver has two different inputs, the signal coming from row decoder and the enable provided by control logic. With regars to the first one, input load is the output capacitance of the row decoder cell, whose output stage is a balanced inverter. Enable signal is supplied by an inverter chain placed into control logic with an output stage, whose transistors are shown in Figure 3.24, characterized by an overall pMOS width of $13.8\,\mu m$ and an overall nMOS width of $4.6\,\mu m$. Output node capacitance is due to bitcell's access transistors. Actually, it is necessary to distinguish read wordline from write wordline since they have a different load. Each read wordline is connected to the gate of a transistor with a channel width of $0.84\,\mu m$, while a write wordline drives two pass transistors with a size of $0.42\,\mu m$.

$$C_{in\_wordline\_driver} = 4.2\,C_{drain_{min}} = 2.94\,fF$$

$$C_{en\_wordline\_driver} = 5\,(\frac{1.38\,\mu m}{0.42\mu m} + \frac{0.46\,\mu m}{0.42\mu m})\,C_{drain_{min}} = 15.3\,fF$$

$$C_{out\_wwl} = 32\,(C_{gate_{min}} + C_{gate_{min}}) = 12.8\,fF$$

$$C_{out\_rwl} = 32\,Cgate_{min} = 12.8\,fF$$

The two possible output capacitances are numerically the same, so it is possible to perform only one simulation in order to characterize the delays

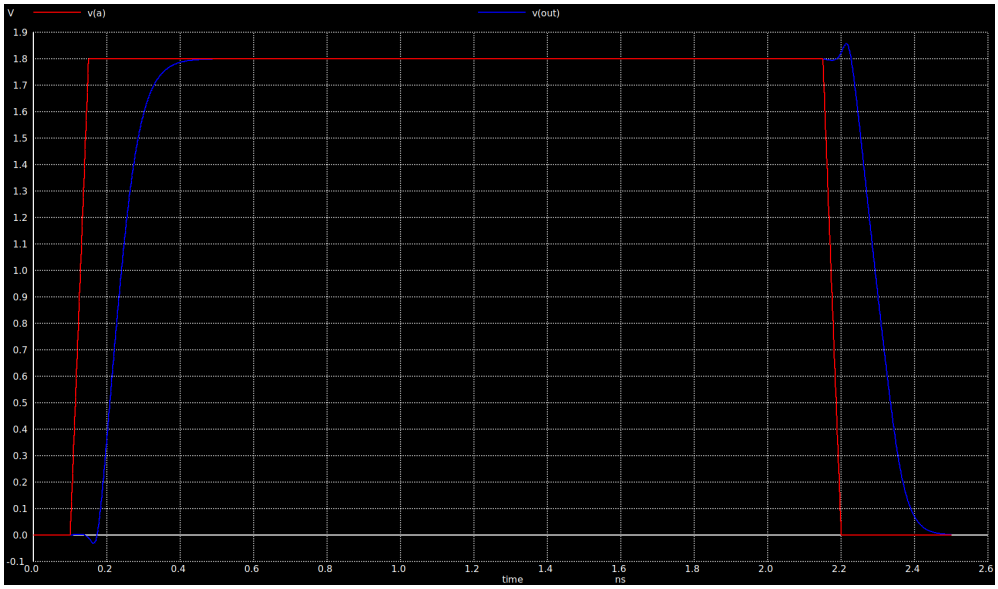


Figure 3.25: Wordline driver simulation.

of the wordline driver.

$$t_{HH_{50}} = 45\,ps$$

$$t_{LL_{50}} = 47\,ps$$

$$t_{rise_{10-90}} = 19\,ps$$

$$t_{fall_{10-90}} = 20\,ps$$

Each wordline driver cell contains three AND 2 gates and input and output pins are all realized by using metal 1 to be correctly interfaced with row decoder outputs and bitcell array's wordlines respectively. Area is equal to $8.29\,\mu m$ x $6.02\,\mu m = 49.9\,\mu m^2$.
After LVS and extraction, the new netlist is simulated. The obtained results are the following ones:

$$t_{HH_{50}} = 93\,ps$$

$$t_{LL_{50}} = 97\,ps$$

$$t_{rise_{10-90}} = 41\,ps$$

$$t_{fall_{10-90}} = 40\,ps$$

During the simulation of the extracted netlist, whose waveforms are a copy

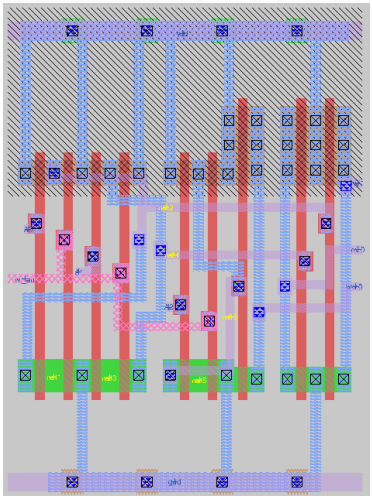


Figure 3.26: Layout of wordline driver cell.

of the ones reported in Figure 3.25, power consumption due to the transitions is measured. Then, leakage power is evaluated as well.

$$P_{tran} = 75\,\mu W$$

$$P_{leakage} = 0.45\,\mu W$$

3.5 D Flip Flop

D flip flop is present inside the Register File to sample the input address, the input data that has to be written inside a specific bitcell and the signals received by control logic. The chosen schematic is depicted in Figure 3.27. It is a Positive Edge Triggered (PET) static D flip flop in which input and output inverters are useful to reduce noise sensibility even though they introduce a delay. Transistors' sizing is aimed to get a balanced flip flop, so all the nMOS have a channel width of $0.42\,\mu m$, while the channel width of all the pMOS is $1.35\,\mu m$.
The chosen topology has two inputs, that are clock signal and a signal provided by the external. Inside control logic, clock signal passes through an inverter chain whose output stage is characterized by a pMOS with 57 fingers

of $1.38\,\mu m$ and nMOS with 57 fingers of $0.5\,\mu m$. The other input load cannot be estimated by considering the other stages, but it is necessary to make an assumption. For instance, it is possible to consider 20 times the minimum drain capacitance. Output load is evaluated by considering that D flip flop is connected either to predecoder (input stage is a balanced inverter) or to write driver.

$$C_{clk\_dff} = 28\,(\frac{1.38\,\mu m}{0.42\mu m} + \frac{0.5\,\mu m}{0.42\mu m})\,C_{drain_{min}} = 87.7\,fF$$

$$C_{din\_dff} = 20\,C_{drain_{min}} = 14\,fF$$

$$C_{out\_row\_addr\_dff} = 4.2\,C_{gate_{min}} = 0.84\,fF$$

$$C_{out\_data\_dff} = 4.2\,C_{gate_{min}} = 0.84\,fF$$

Simulation shows the expected behaviour and it makes possible to measure the delay introduced by this module, setup time and hold time with the following results:

$$t_{CLK->Q} = 135\,ps$$

$$t_{setup} = 102.5\,ps$$

$$t_{hold} = -56\,ps$$

Setup and hold times are measured by following a procedure based on monitoring the value of $t_{CLK->Q}$.
Concerning setup time, the first step consists in measuring $t_{CLK->Q}$ when the input signal precedes the active edge of clock signal by enough time. Then, their time distance has to be reduced until clock to output delay becomes 5% higher than the first measured value. At that point, the time between the input signal and the active edge of clock is setup time.
Hold time measure follows the same strategy. The value of $t_{CLK->Q}$ is measured when a transition of the input signal is delayed by an enough wide time interval with respect to the active edge of clock. Once again, time distance is reduced until $t_{CLK->Q}$ becomes 5% higher than the first measured value. In this case, the time between the two signals is hold time.
The fact that measured hold time is negative means that metastability caused

by a too fast path could never happen.

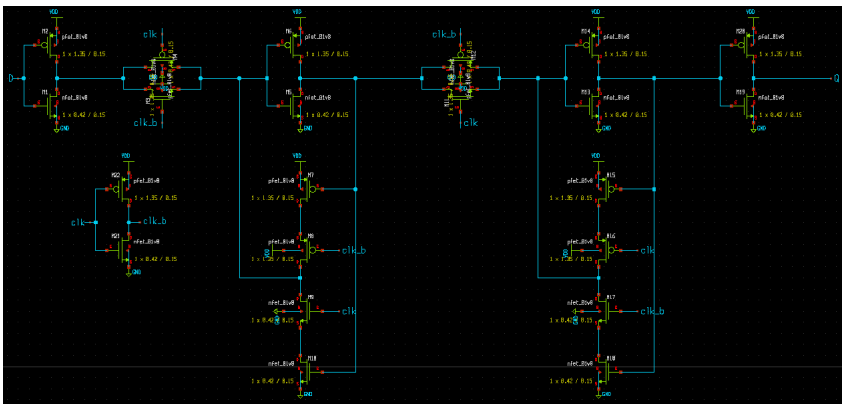


Figure 3.27: Schematic of D flip flop.

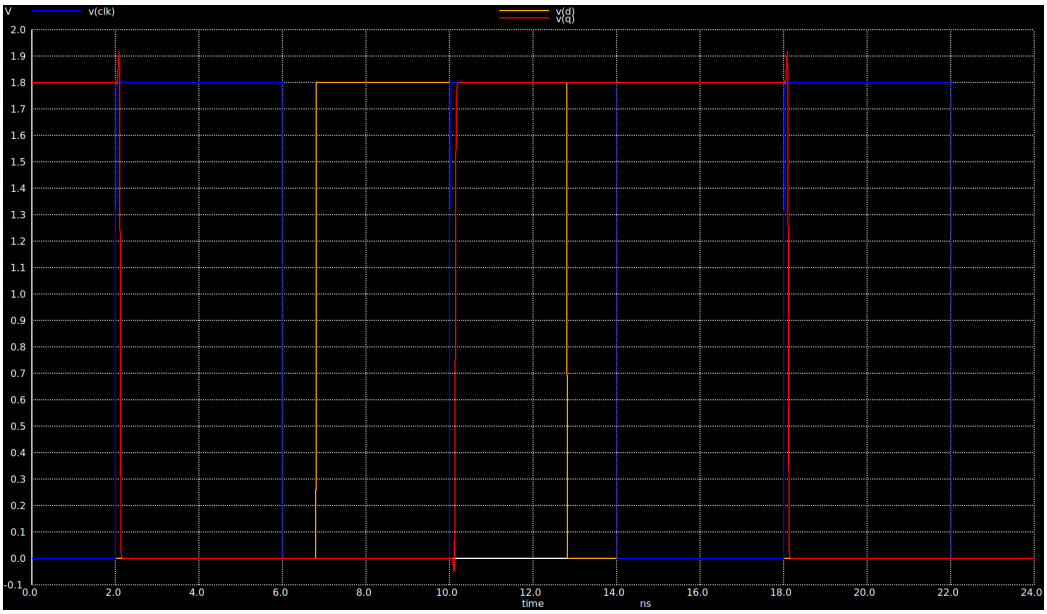


Figure 3.28: Simulation of D flip flop.

The layout is done by considering the constraint about input and output pin's layer, that must be metal 2 in order to get an interface compatible with control logic which provides clock signal, write driver which receives the data sampled from the external and predecoders that receive the sampled input address. Moreover, in the design of this module a special attention is given to area since it has many transistors. If layout is not realized in a proper

way, there is the risk to obtain a module with a very wide area. The area of this D flip flop is equal to $4.79\,\mu m$ x $7.47\,\mu m = 35.78\,\mu m^2$, 40% smaller than the one present in the preexisting Sky130 D flip flop present in previous version of OpenRAM.
Once obtained the extracted netlist, it is simulated. The measured timing parameters are:

$$t_{CLK->Q} = 205\,ps$$

$$t_{setup} = 120\,ps$$

$$t_{hold} = -53\,ps$$

Finally, power estimation is performed during extracted netlist simulation.

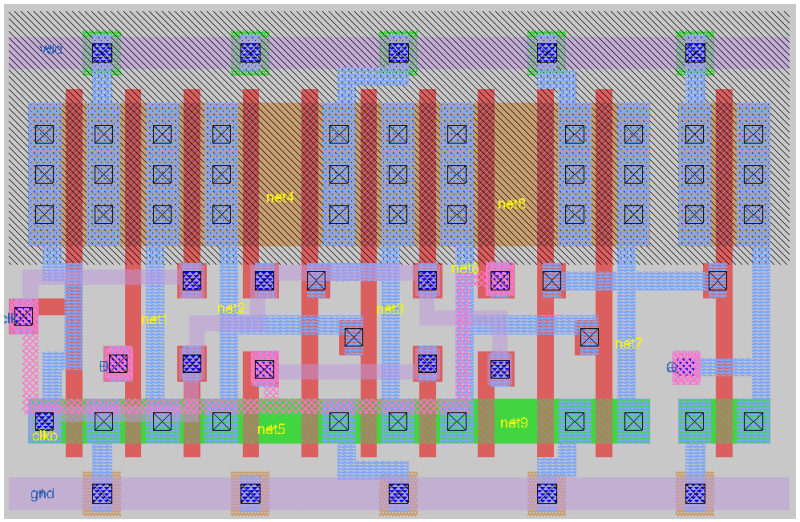


Figure 3.29: Layout of D flip flop.

Both the samplings of a 0 and a 1 are considered. Clock period is fixed to $4\,ns$.

$$P_{sample\_0} = 16.4\,\mu W$$

$$P_{sample\_1} = 20.2\,\mu W$$

$$P_{leakage} = 1.6\,\mu W$$

Chapter 4

Framework extensions

The generation of the $2R/1W$ Register File requires the integration of some Python code to the preexisting one. OpenRAM doesn't natively support the creation of a memory based on the bitcell shown in Section 3.2, characterized by separated read and write ports that are implemented through a simple bitline without the complementary one. OpenRAM's framework can generate memories with multiple ports, but it supports only bitcells with complementary bitlines.

Given the tool's organization description presented in Section 1.2.2, a more detailed overview about the way the tool is built is provided. When OpenRAM is downloaded from Github repository [12], the folder contains two main subfolders that are compiler and technology. Compiler folder includes all is needed to create and characterize the array apart PDK, which is contained inside technology folder. Each PDK folder includes the files needed to use a custom module in OpenRAM, so SPICE and GDS files, the SPICE models of the PDK's devices and a subfolder called tech. The last content is very important because here is located the file *tech.py* which incorporates all the design rules, the indications about the sizes of the custom modules' transistors, the features of the transistors such as gate and diffusion capacitances, all the analytical delay and power parameters useful during characterization and logical effort parameters of hand-made cells. Compiler contains many subfolders, so in the following the most important ones will be mentioned.

Base is a folder including all the files that constitute the basis of the tool

since they perform the creation of the memory layout, the creation of the circuit level description and the definition of the cells and the layers that are used inside the memory. Bitcells folder includes the definition of all the possible bitcells that can be used in the array. In each file a class is created and each class calls another class defined in the file *bitcell_base.py*. It contains some functions that declare the pin names of the bitcells and other that are useful during simulation and parasitic extraction. Custom subfolders comprises one file per each custom module providing the creation of the respective class. Moreover, each file contains few routines used during characterization. Pgates subfolder includes the description of the gates and modules that are created through OpenRAM's framework by starting from the basic transistors' models. Each file contains not only what is needed for characterization, but also the function in which the transistors are instantiated, placed and routed in the proper way. Those functions take advantadge of other functions defined in the tool's basis scripts. Modules folder is fundamental since it comprises the code to create all the modules that compose the memory. Every file defines a class, in which there are some functions with a specific aim such as defining the pins of the module, instantiating the internal blocks (custom and/or pgates), placing and routing them to create the target module, creating the layout pins in a certain position by using a specific layer. Sram folder contains the classes that are called by top level script to create the memory array. Those classes are able to create the entire memory by calling the lower level classes which in turn call the lower level class and so on. The memory is built by following a top-down methodology. Every module is created thanks to a class called *sram_factory*. Top level script, which can be found in compiler folder, is *openram.py*. It gives start to the generation of the memory and it calls the file *globals.py*, which parses all the arguments and performs the global OpenRAM setup as well. Finally, a very important class to mention is defined in *options.py*. It holds all of the OpenRAM options. All of those options can be overridden in a configuration file that is the single required command-line positional argument for *openram.py*, as discussed in Section 1.2.3.

In this work, the preexisting framework is not modified, so it is possible to exploit all the functionalities of OpenRAM for the creation of a single

port memory. This project aims to integrate a functionality consisting in creating a multiported memory. In order to do so, new code is created along the same lines as the preexisting one. It is necessary to organize a strategy to differentiate the creation of a RF and the creation of a single port memory, and this task is performed by introducing a boolean variable called *RF_mode*, set in configuration file. If *RF_mode* has value False, then the native OpenRAM's code is run, while if it has value True, the code integrated in the setting of this project is executed. Obviously, when the number of read ports and write ports are respectively set to 2 and 1, *RF_mode* must be True.
In the following, the additions brought to the compiler will be explained in details. Besides the operations on the compiler, also the characterizer has been faced. Some parts of the entire code has been adapted to support the characterization of such a memory without any significant addition. The difference between the preexisting code and the new one regard only the label of the pins.

4.1 Custom module integration

The integration of an hand-made module requires a series of steps.

1. Design of the module.

2. Obtaining circuit level (SPICE file) and graphical level (GDS file) descriptions of the module. The former is acquired from the schematic, the latter by executing the command *gds write* on Magic console once the layout is completed. Before creating GDS file it is very important to introduce a property with the name *FIXED_BBOX* {} to identify the boundary box of the module, otherwise OpenRAM will not be able to place it. Between the curly brackets of that property it is necessary to specify the four coordinates to recognize the box's sides. They can be obtained by typing the command *view_bbox* on Magic Console. It must be considered that OpenRAM will identify the third and the fourth coordinates as the width and the height of the cell.

3. Copy of the files in the right subfolder of technology folder.

4. Creation of a new Python script that defines a new class with the name of the module. It has to contain the informations about the module useful during characterization. Moreover, the pins must be defined with the correct names and order. It can be realized along the same lines as a preexisting script describing a custom modules.

5. Addition of the new file either to custom subfolder or bitcells subfolder inside compiler folder.

6. Modification of *custom_cell_properties.py* file that can be found in base subfolder. It is necessary to add an attribute to the class *cell_properties* with the name of the module and its pins by specifying the pin type. It is very important that the name and the order of the pins is the same of the SPICE file.

7. Definition, instantiation, placing and routing of the custom cell at the level of the script describing the module in which the cell itself is used.

It is possible to mention what are the procedures to perform the last step:

- Definition. A sub-module used inside an higher hierarchy level module is defined in the function *add_modules*. The particular routine, *self.factory.create(module_type = "module_name")*, calls the *sram_factory* class.

- Instantiation. Once the sub-module is created, it has to be instantiated through the function *self.create_"module_name"* in which the instance is added thanks to the call of the routine *add_inst(name, module_name)*. Then, a list containing the names of the sub-module's pins is built and finally the routine *self.connect_inst(list_of_pins)* is called.

- Placing. The function used to place a sub-module is *self."module_instance".place(offset)* where offset is the relative position of the sub-module inside the higher level block. The routine *place* is defined in *hierarchy_layout* class. An important aspect in placing the modules is the possibility to share power supply and ground rails, as shown in

Figure 4.1. This is possible by setting the boundary coordinates in such a way an half of the two rails is excluded from the box. Moreover, in order to achieve that result, the cells need to be mirrored with respect to X axis. It is possibile thanks to an input variable of the function *place*.

- Routing. It is an operation that OpenRAM performs by means of a function named *route_"module_name"*. They call *add_path("metal_layer", [points' coordinates])* and *add_via_stack_center(from_layer=, to_layer=, offset=)*. The first one is a routine that creates a metal line between two points. Its shape depends on the number of points given in input to the routine. With two points the line is straight, with three there will be a connection between the first point and the second point, and then a connection between second and third point, and so on. The second routine creates a via from a metal layer to another one by using the proper layer stack among the ones defined in the file *tech.py*.

4.2 Parametric modules creation

The procedure to realize a parametric module is the same with respect to the one just described for custom module with the difference that no SPICE or GDS files have to be created and the cells that are instantiated are pMOS and/or nMOS. There are three modules of the 2R/1W Register File that have been realized by following this approach. They are column multiplexer, precharge circuit and control logic, produced by starting from the preexisting scripts describing those components used in the single port memory implementation.

4.2.1 Column multiplexer

OpenRAM implementation designed in this work introduces column multiplexing when an array's row contains more than one word. Column multiplexer cell is composed by three pass transistors, one per each port, that are connected to the cell's bitlines from a side and to the bitline sensing circuits'

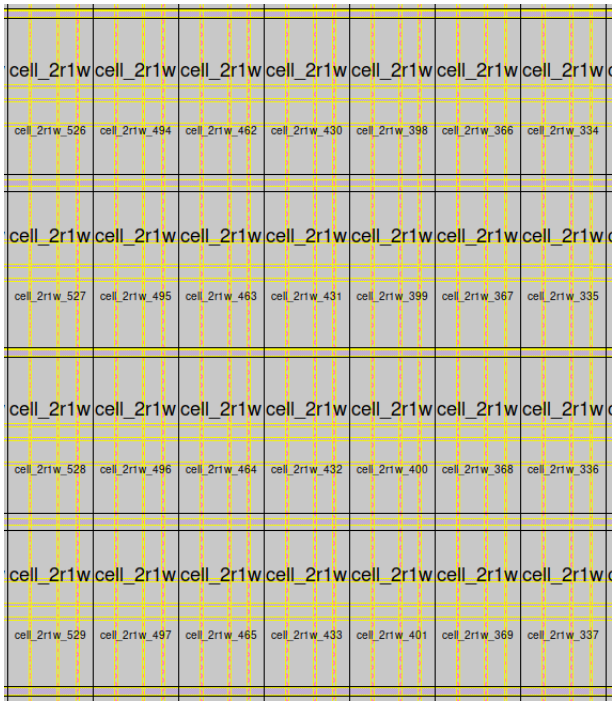


Figure 4.1: Shared power and ground rails between bitcells in a 32 x 32 2R/1W Register File.

inputs or write driver's output from the other side. They are driven by the column decoder, whose constituting module depends on the number of bits needed to decode the column. In other words, in order to activate the column mux's transistors in the correct way, the used module depends on the number of words per row. If column address length is 1, then the decoder is simply an inverter, otherwise it is a predecoder with an appropriate number of inputs.

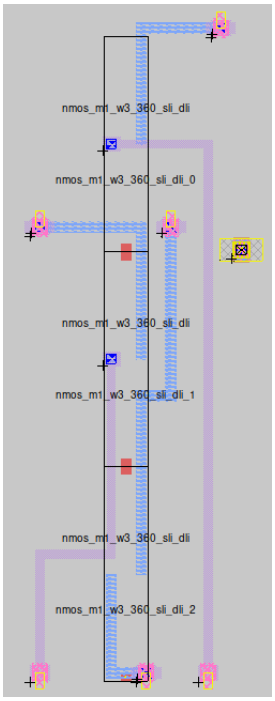
From layout point of view, column multiplexer cell is depicted in Figure 4.2a, it consists of three n-channel MOSFETs placed one above the other. There is a ground contact placed to bias the substrate in the correct way. The total width of the cell is equal to the bitcell's width, so the column multiplexer and the bitcell array have the same total width.

4.2.2 Read bitlines precharge circuit

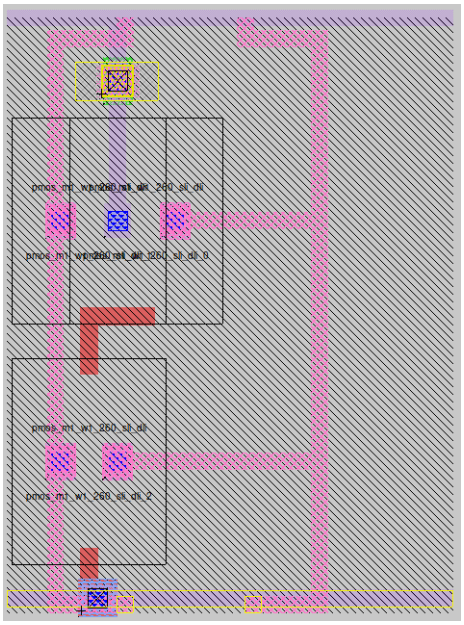
The bitline conditioning circuit used in this project has a topology that follows the one shown in Figure 2.21c with 3 pMOS driven by a clocked signal. In this case, the signal is provided by control logic and its activation causes the precharge of both read bitlines. The layout of the single cell is shown in Figure 4.2b. Two devices share the diffusion connected to power supply, while the other two diffusions are connected to read bitlines. Then, the other pMOS is driven by the control signal and has source and drain connected to bitlines. All the transistors have a width that is the minimum one times the ratio between electrons and holes mobility.
The metal 2 have that pattern in order to get a straight connection to the bitcell's bitlines. In other words, the distance between bitlines and between the bitlines and the borders at the top and at the bottom of the precharge cell are equal to the bitcell's ones. Once again, the width of the cell is equal to the bitcell's width.

4.2.3 Control logic

Control logic is a fundamental module that guarantees the correct operation of the entire memory by giving the control signals in the right moment. In order to do so, it receives some external signals that pass through sequential and combinational stages to produce control signals.
In this project, control logic receives write enable and chip select, that are both active low, from the external. Each of them is the input of a buffered D flip flop, so a D flip flop followed by a chain of two inverters. Thus, that block is able to produce both the input signal and its complementary. As it is possible to see from Figure 4.3, control logic contains 6 combinational blocks placed one above the other. Moreover, there are 6 metal 2 vertical rails that carry some signals with a particular meaning. 4 of them are the outputs of the two buffered D flip flops. The other two lines are related to clock. The clock signal provided by the external is transferred to one of those rails and finally there is a signal derived from clock produced by a combinational gate that receives also the chip select provided by a buffered DFF. This block is an inverter followed by a 2-inputs AND gate, and the



(a) Layout of column multiplexer cell.



(b) Layout of read bitlines precharge circuit.

Figure 4.2: Column multiplexer and precharge circuit.

first stage provokes the inversion of the external clock. It means that at the output it is possible to find the complementary of the delayed external clock when chip select is active. The output is then transferred to a vertical metal 2 rail to be distributed to the other combinational gates of the control logic. Control logic has 4 output signals: wordline enable, precharge enable, write enable and buffered clock ordered from the top to the bottom of Figure 4.3. Wordline enable is produced by a pdriver, a module that is composed by a certain number of inverters sized to drive a load. In this case, *pdriver* receives the complementary buffered clock that passes through a chain of four inverters. Precharge enable is produced thanks to a *pdriver* containing two inverters. A 2 inputs AND gate receiving the complementary buffered clock and one of the outputs of a buffered D flip flop produces write enable. Finally, buffered clock that is sent to the rest of the memory is provided by means of a *pdriver* that includes a chain of four inverters and that receives

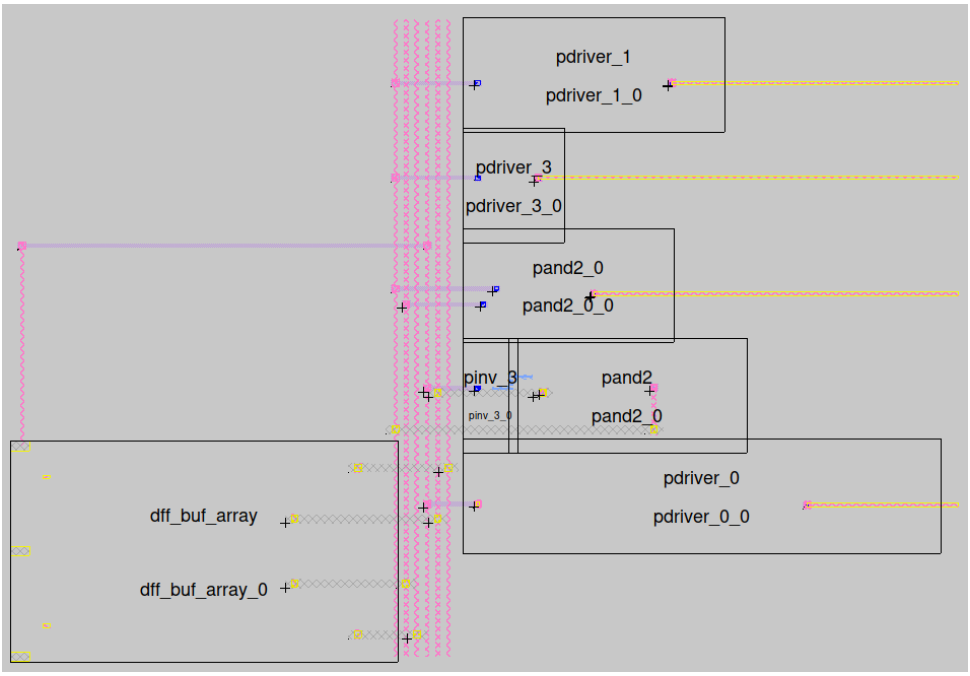


Figure 4.3: Block diagram of control logic.

the external clock.
All the blocks of control logic were part of the framework before the starting point of this project, so all the issues about gate sizing and speed haven't been faced in this work.

Chapter 5

Register File implementation

Once the compilation is launched, OpenRAM manages to create the entire memory by using all the modules that have been analyzed until now. A complete view of the $2R/1W$ Register File is provided in Figure 5.1. It is possible to notice the highest level modules that compose the entire memory, so bank, control logic, row address D flip flop array and data D flip flop array. Bank is in turn constitute by port data, containing all the column circuitry, port address, including all the row circuitry and bitcell array.
A very important aspect to analyze regards the adopted method to allow simultaneous operations on the memory bitcells, a fundamental feature of a Register File used in processors. The memory is designed in such a way that there are three separated row addresses, one per each port, that are sampled by three different arrays of D flip flops. Then, the sampled bits are the inputs of the predecoders, whose number is three times the number of predecoders used in a single port memory implementation. The outputs of the three predecoding blocks become the input of the row decoder cell with 2 or 3 inputs, depending on address bits, designed in Section 3.4. It is necessary to be very careful in connecting the predecoders outputs to row decoder's AND gates inputs because the three address are independent and they haven't to be mixed as inputs of the same AND gate.
Obviously, there is an area penalty due to the introduction of two D flip flop arrays and of two predecoding modules, but it is only the consequence of improving the performance of the memory. Hence, as said in Chapter

2, the most important features of a memory are strictly related, if one of them is enhanced, then another one is worsened. In this case, area penalty depends on the number of bits used for address. For instance, by considering a 32 x 32 Register File, OpenRAM provides the information about area after compilation, that is roughly equal to $108000\,\mu m^2$. As reported in Section 3.5, the produced D flip flop has an area of $35.78\,\mu m^2$, so in a 32x 32 RF, where row address is codified in 5 bits, there is an overhead of 10 D flip flops. Moreover, it is necessary to consider also the introduction of two $2:4$ and two $3:8$ predecoders, which respectively have an area of $162\,\mu m^2$ and $387\,\mu m^2$. Area overhead is evaluated as:

$$A_{overhead} = 10 \cdot 35.78\,\mu m^2 + 2\,(162 + 387)\,\mu m^2 = 1455.8\,\mu m^2$$

$$A_{overhead\_\%} = \frac{1455.8}{108000 - 1455.8} \cdot 100 = 1.366\,\%$$

It is possible to state that a great advantage in performance is achieved at the cost of only $1.366\,\%$ area overhead.
Finally, another important point to highlight is the memory timing derived from the internal control logic connections discussed in Section 4.2.3. The precharge enable and the wordline enable are generated by a delayed inverted replica of clock signal. It means that wordlines are active when clock has a low voltage value, so read and write operations can be performed in the second part of the cycle, while read bitlines are precharged in the first part of the cycle, so when precharge enable is low. This strategy leads to avoid any kind of conflict on read bitline between the cell's transistors that offer a path to discharge the bitline and the pMOS of precharge circuit. Write enable is originated from inverted clock as well, so the input data is transferred on write bitline in the second part of the clock period. The first part of the cycle is also used to generate the row and column addresses. Under these conditions, the reading operation provides for a situation in which the data stored in a bitcell is valid at the output of the memory in the second half of the clock period in which the read address is sampled.

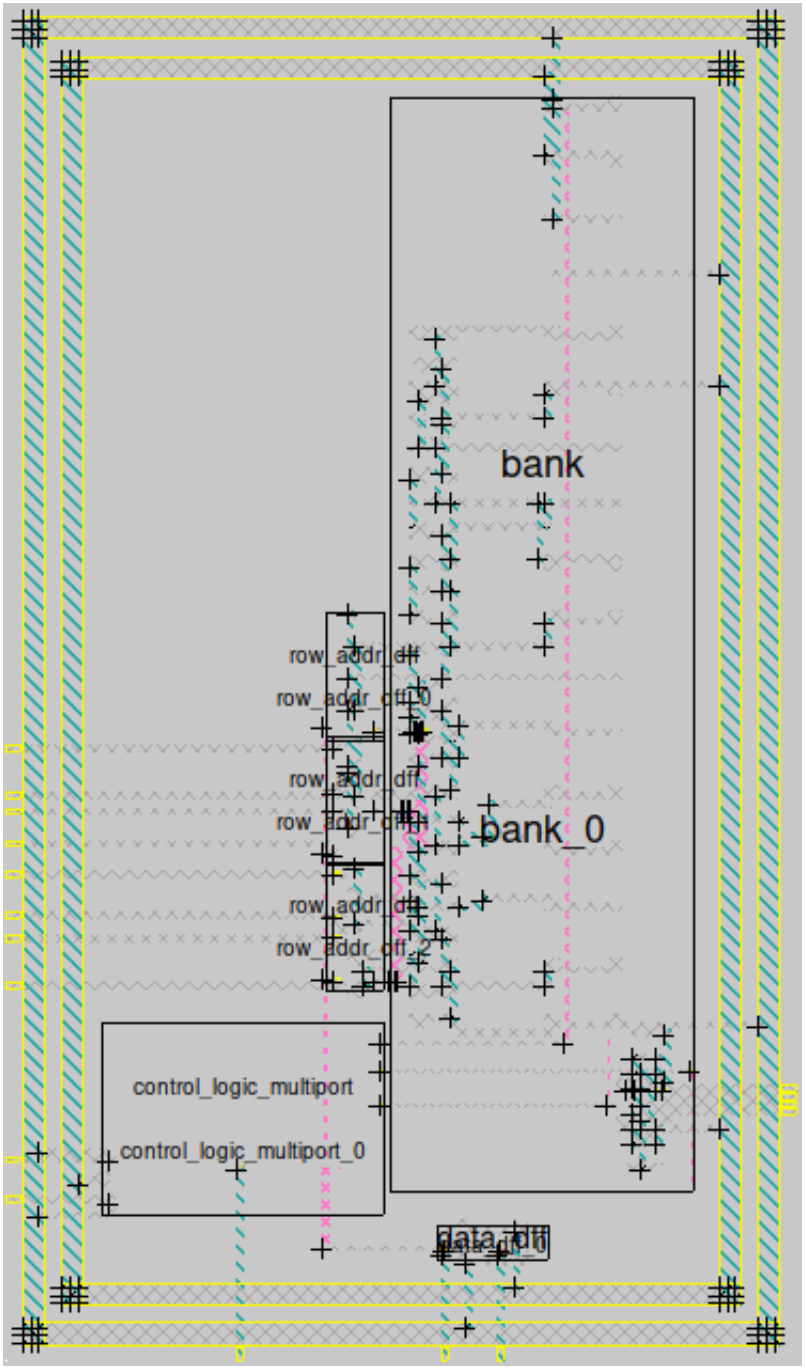


Figure 5.1: Highest level view of the Register File.

Chapter 6

Simulation and results

This chapter contains the results of the simulations performed in order to ensure that what has been designed works properly. For this purpose, two simulations are executed. The first one is focused on checking if the expected functionalities are achieved by testing only one bitcell. Then, a 32 x 32 Register File is tested in order to evaluate its features from performance and power point of view in different process corners and with different environmental conditions.

6.1 Simulation of one bitcell

The simulation is conducted by setting up a schematic, shown Figure 6.1, containing the modules used inside the $2R/1W$ Register File, including the non custom cells as well. Only a bitcell is present inside the scheme, so there are three possible wordlines that can be activated. It means that 3 bits are needed for the row address to allow simultaneous operations on the bitcell. The predecoders provided by OpenRAM framework and the row decoder cells designed in Section 3.4 are not needed in this case, so row address flops' outputs are the inputs of the wordline driver. This last module receives also the enable provided by control logic. Each control logic's block is present inside the schematic with the topology and the transistors' sizing established by OpenRAM compilation.
The input and output signals in the simulation scheme are the address bits,

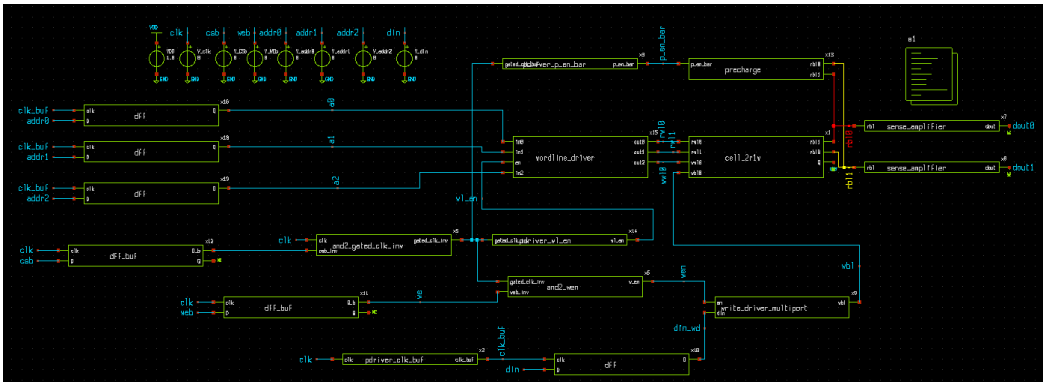


Figure 6.1: Schematic used for the simulation of the memory with one bitcell.

the input data, the clock signal, write enable, chip select and output data. In order to achieve the simulation aim, the first step consists in setting clock period to $4\,ns$. Then, a writing of a 1 is performed, followed by a simultaneous reading from both read ports. Thereafter, a 0 is written in the bitcell and then read from both ports. This sequence of operations is obtained by providing the control signals in the proper way. For any operation, chip select must be active, otherwise wordline cannot be activated since wordline enable is a buffered version of the AND between inverted chip select and inverted clock. For the write operation, it is necessary to give in input the activated write enable, the correct address and the input data. When those signals are sampled, in the second half of that clock cycle, write wordline is activated and the signal on write bitline can be written in the bitcell. For reading, it is enough to provide only the address and waiting for the output data according to the explanation presented in Chapter 5. The results of the simulation are shown in Figure 6.2. The signals are ordered in the following way from top to bottom: clock, address0 (referring to first read port), address1 (referring to second read port), address2 (referring to write port), chip select, write enable, input data, write bitline, wordline enable, stored data, write wordline, read wordline of first port, read wordline of second port, output data of first port, output data of second port.

Next step provides for the measure of the delays between wordline activation and output data rising in case of reading and variation of stored data in case of writing. In this case, the SPICE netlist used for simulation doesn't include

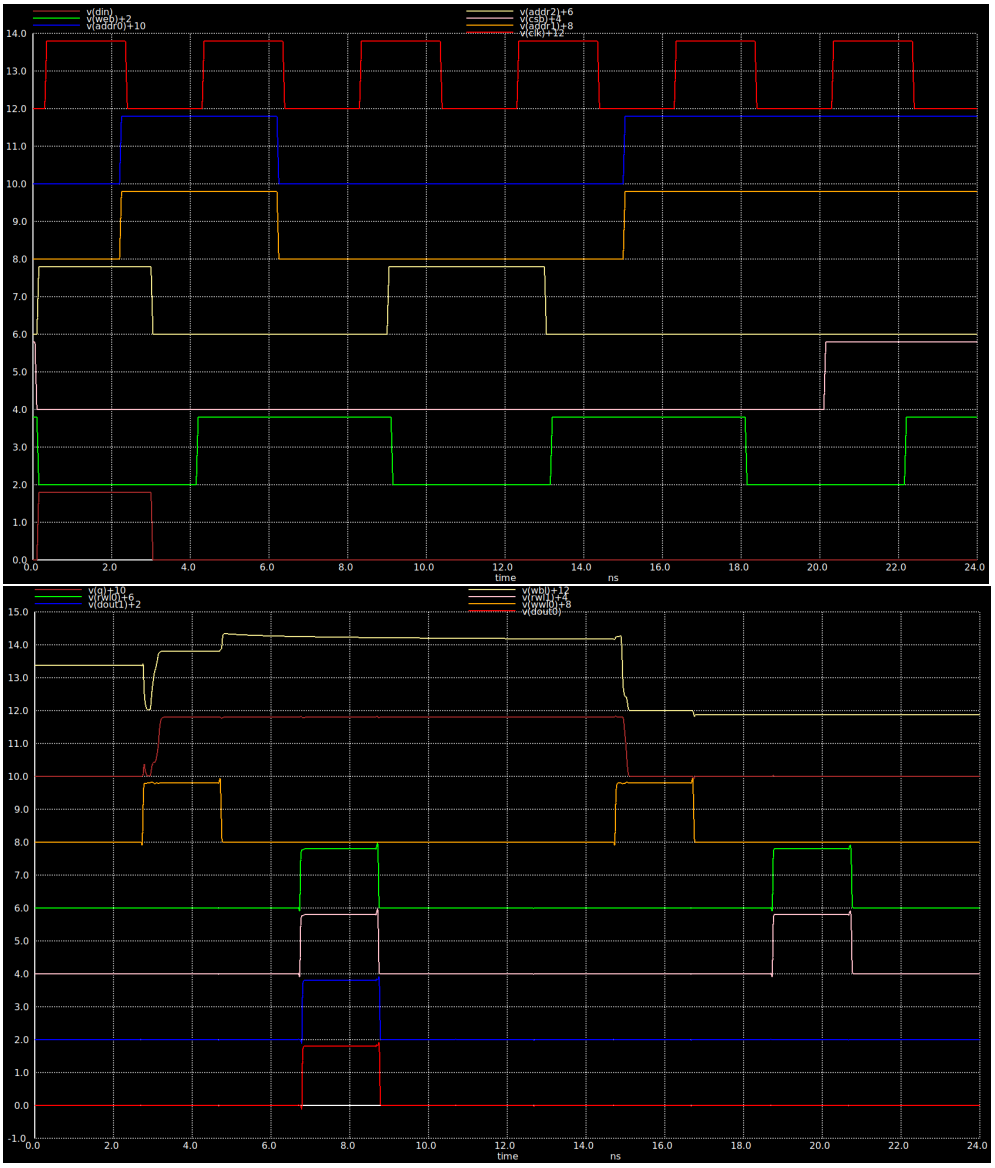


Figure 6.2: Waveforms of one bitcell scheme's simulation.

parasitics. The obtained values, considered at 50 % of the maximum level, are:

$$t_{rwl2dout} = 50\,ps$$

$$t_{wwl2Q\_rise} = 384\,ps$$

$$t_{wwl2Q\_fall} = 287\,ps$$

Moreover, another measure involves the minimum clock period sustained by this scheme. The found value is about $1\,ns$. It means that the maximum operating frequency for this configuration is more or less $1\,GHz$, so $2\,GHz$ as bandwidth in reading.

6.2 32 x 32 Register File simulation

The second simulation regards an entire $2R/1W$ Register File with 32 words of 32 bits. In this case, all the modules are present in the simulation schematic, including predecoders and row decoder cells. The schematic, shown in Figure 6.3, follows the block diagram depicted in Figure 5.1. By considering Figure 6.3, on the top left it is possible to find the three arrays, one per each port, of D flip flop that sample the input addresses. By moving towards the right, the row decoder receives the outputs of row address DFF and produces the inputs of wordline driver array, that in turn provides the wordlines to bitcell array. Bitcell array receives also the write bitlines, supplied by write driver array that can be found on the bottom of the picture. Write drivers' input data are provided by data D flip flop array, placed on the left of write driver array. The outputs of bitcell array are the read bitlines, that are connected to precharge array and to the inputs of bitline sensing circuit array, which produces output data. On the bottom left it is possible to find the control logic.
The parasitic bitline and wordline capacitances due to the non ideality of the metal wires are also considered for the evaluation of the RF performance. In order to estimate their value, some considerations about the bitcell array dimensions and the technological parameters of Sky130 PDK are required. Bitcell array's width is equal to $124.5\,\mu m$, while the height is equal

to $246.5\,\mu m$. The capacitance per area unit of metal 1, used for wordlines, and metal 2, used for bitlines, is $0.134\,\frac{fF}{\mu m^2}$ in both cases. Inside each Register File implemented in OpenRAM by using Skywater130 technology the width of each wire is the minimum possible one, so $0.14\,\mu m$. Wordlines and bitlines capacitance are:

$$C_{wordline} = 0.134\frac{fF}{\mu m^2} \cdot 0.14\,\mu m \cdot 124.5\,\mu m = 2.34\,fF$$

$$C_{bitline} = 0.134\frac{fF}{\mu m^2} \cdot 0.14\,\mu m \cdot 246.5\,\mu m = 4.63\,fF$$

The simulation provides for a writing of a 32 bits data on the top row and then a simultaneous reading from both read ports. The waveforms, obtained by setting clock period to $4\,ns$, are shown in Figure 6.4 and are ordered in the following way from top to bottom: clock, chip select, write enable, write address (five bits), least significant input bit, most significant input bit, address of first read port (the one of second port is identical), least significant output bit on first read port, most significant output bit on first read port, least significant output bit on second read port, most significant output bit on second read port.

The key point of this test concerns the evaluation of memory performance and power dissipation by considering parasitic parameters. Several tests have been executed in order to characterize the memory in the 5 different process corners whose models are provided by SkyWater130 PDK, and in different environmental conditions by changing temperature and power supply value. Clock period is fixed to $8\,ns$.

The measured delays concerns the time between the activation of a read wordline and the rising of the output data and the times between the activation of the write wordline and the rising and falling of the stored data. Once again, the delays refer to 50 % of the final value of each signal after the transition.

Those delays are taken into account since they are the most critical ones in terms of memory performance. As explained in Chapter 5, read and write procedures occur in the second part of the cycle, while in the first part the input signals are sampled and then they pass through the row decoding mod-

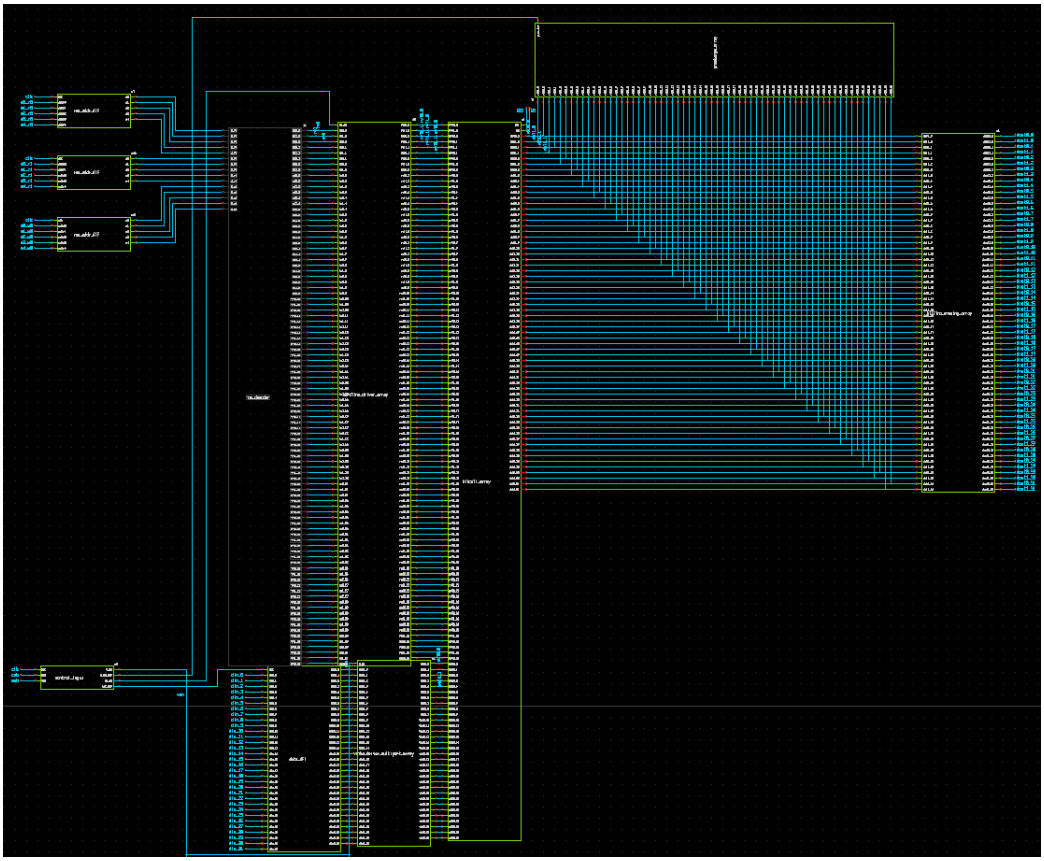


Figure 6.3: Schematic of the 32 x 32 RF simulation.

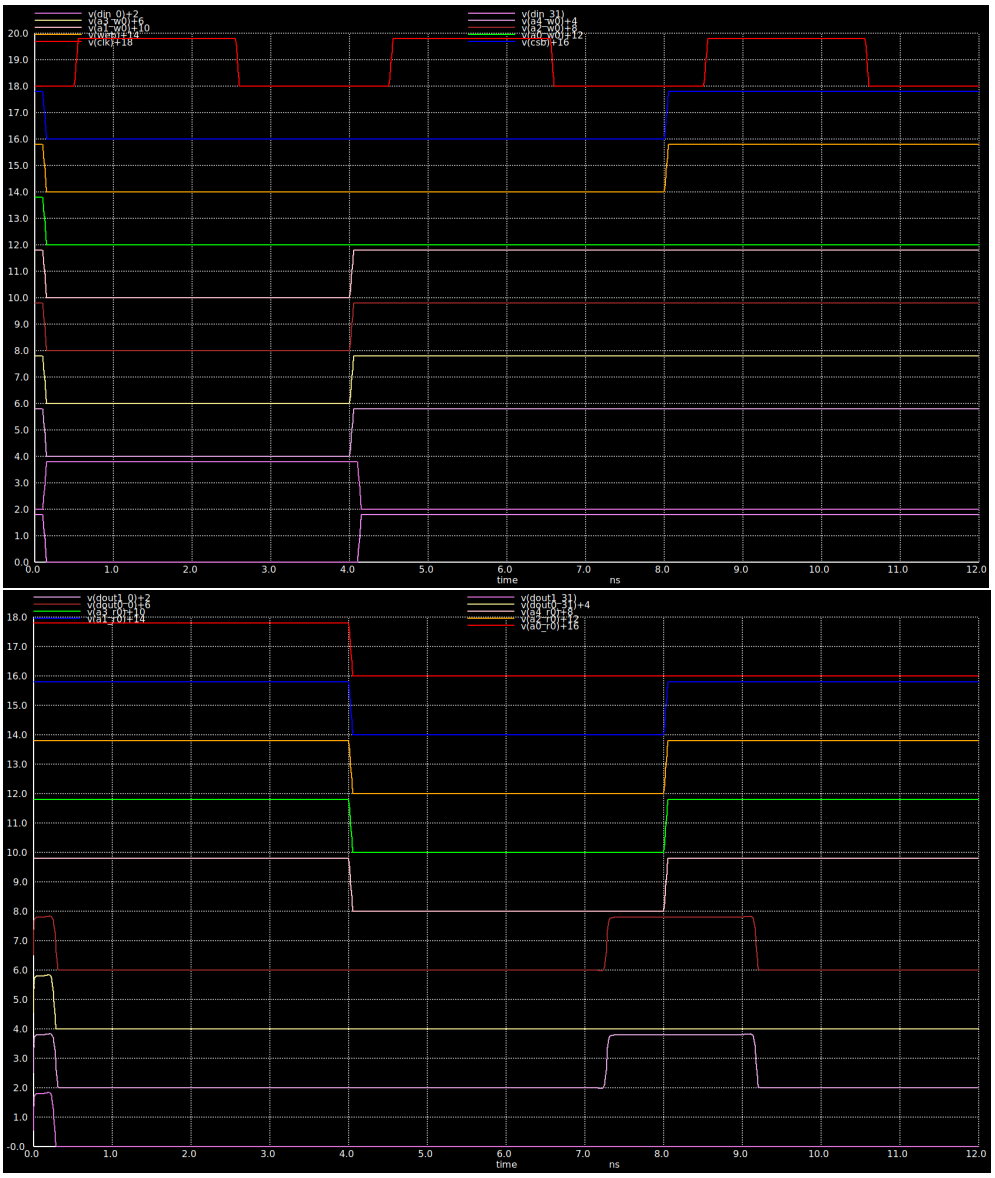


Figure 6.4: Simulation waveforms of the 32 x 32 Register File.

ules path or the modules useful for writing path. Simulations showed that the second part of the cycle is the most critical one regarding timing requirements, so the measured delays can be seen as an indication of the maximum operating frequency of the memory.
Table 6.1 shows the timing features of the memory in the 5 process corners provided by SkyWater130 PDK. The simulations are performed with a temperature of $27\,°C$ and a power supply voltage of $1.8\,V$, so in a standard condition.

Corner	$t_{rwl2dout}$	t_{wwl2Q_rise}	t_{wwl2Q_fall}	f_{max}
TT	$530\,ps$	$466\,ps$	$453\,ps$	$820\,MHz$
FF	$396\,ps$	$341\,ps$	$364\,ps$	$910\,MHz$
SS	$778\,ps$	$1.08\,ns$	$610\,ps$	$400\,MHz$
FS	$566\,ps$	$902\,ps$	$670\,ps$	$500\,MHz$
SF	$545\,p$	$588\,ns$	$462\,ps$	$770\,MHz$

Table 6.1: Measured delays and maximum operating frequency of the 32 x 32 Register File in 5 different process corners.

Concerning power, there are 5 different measurements that account for reading and writing of a 0 or a 1 and leakage power. The evaluations are intended as medium power over a certain interval of time. Table 6.2 shows the power features of the memory in the 5 process corners provided by SkyWater130 PDK. Once again the simulations are performed with a temperature of $27\,°C$ and a power supply voltage of $1.8\,V$. The results about power consumption during reading refer to only one port.

Corner	P_{read_0}	P_{write_0}	P_{read_1}	P_{write_1}	$P_{leakage}$
TT	$0.99\,mW$	$2.36\,mW$	$1.14\,mW$	$5.70\,mW$	$1.41\,mW$
FF	$1.5\,mW$	$3.40\,mW$	$1.68\,mW$	$9.10\,mW$	$108.23\,mW$
SS	$0.98\,mW$	$2.16\,mW$	$1.00\,mW$	$3.58\,mW$	$0.68\,\mu W$
FS	$1.00\,mW$	$2.48\,mW$	$1.76\,mW$	$4.97\,mW$	$0.13\,mW$
SF	$1.00\,mW$	$2.38\,mW$	$1.95\,mW$	$9.91\,mW$	$2.9\,mW$

Table 6.2: Measured leakage and medium power during reading and writing of the 32 x 32 Register File in 5 different process corners.

Given these two tables, it is possible to consider the strict relation between

Temperature	$t_{rwl2dout}$	t_{wwl2Q_rise}	t_{wwl2Q_fall}	f_{max}
$0\,°C$	$559\,ps$	$504\,ps$	$413\,ps$	$770\,MHz$
$27\,°C$	$530\,ps$	$466\,ps$	$453\,ps$	$820\,MHz$
$50\,°C$	$527\,ps$	$477\,ps$	$473\,ps$	$820\,MHz$
$70\,°C$	$557\,ps$	$565\,ps$	$515\,ps$	$760\,MHz$

Table 6.3: Measured delays and maximum operating frequency of the 32 x 32 Register File working with 4 different temperature values.

performance and power. In correspondence of the FF corner, the Register File is able to work with an higher frequency than the normal one evaluated in TT corner, while it is slower in SS corner. This is something expected since FF corner is characterized by a lower threshold voltage and an higher value of β than TT corner, while SS accounts for the opposite conditions. The variation in speed is reflected in power features. The higher is the speed, the higher is the power consumption.
This concept is valid for FS and SF corners as well. SF corner is faster than FS one, therefore power dissipation is higher. The last result leads to declare that p-channel MOSFETs have an higher relevance on improving performance and increasing power dissipation of the entire Register File.
A very important point to highlight is the value of leakage power. In FF corner it is almost 100 times higher than in TT corner, while in SS it is 3 orders of magnitude lower. This wide variation can be justified by considering the exponential relation between threshold voltage and subthreshold current.
The same parameters have been evaluated by keeping the focus on the environmental condition. The Register File is simulated in with different operating temperature and power supply voltage. Concerning temperature, 4 different points in the range $0 - 70\,°C$ are considered, while power supply is set in such a way there is a $\pm 10\,\%$ of displacement with respect to its nominal value. All the results are shown in the following tables. The chosen process corner in these cases is TT.
As a consequence of temperature variation, the Register File shows slight changes in timing parameters and power parameters with respect to nominal temperature of $27\,°C$. As the temperature increases from the nominal value, the leakage power increases as well because threshold voltage drop. Another

Temperature	P_{read_0}	P_{write_0}	P_{read_1}	P_{write_1}	$P_{leakage}$
$0\,°C$	$0.97\,mW$	$2.33\,mW$	$1.12\,mW$	$4.75\,mW$	$0.32\,mW$
$27\,°C$	$0.99\,mW$	$2.36\,mW$	$1.14\,mW$	$5.70\,mW$	$1.41\,mW$
$50\,°C$	$1.15\,mW$	$2.34\,mW$	$1.14\,mW$	$5.20\,mW$	$2.38\,mW$
$70\,°C$	$0.96\,mW$	$2.40\,mW$	$1.27\,mW$	$8.03\,mW$	$4.11\,mW$

Table 6.4: Measured leakage and medium power during reading and writing of the 32 x 32 Register File working with 4 different temperature values.

V_{DD}	$t_{rwl2dout}$	t_{wwl2Q_rise}	t_{wwl2Q_fall}	f_{max}
$1.62\,V$	$694\,ps$	$651\,ps$	$638\,ps$	$625\,MHz$
$1.8\,V$	$530\,ps$	$588\,ps$	$453\,ps$	$820\,MHz$
$1.98\,V$	$467\,ps$	$436\,ps$	$393\,ps$	$910\,MHz$

Table 6.5: Measured delays and maximum operating frequency of the 32 x 32 Register File working with 3 power supply voltage values.

effect that must be consider is the drop of the carriers mobility with the increasing temperature. It is the reason why the performance of the memory doesn't improve at high temperature.
Power supply variation brings to expected results. As the supply voltage increases, delays are reduced and power dissipation increases. With a voltage of $1.98\,V$ it is possible to reach the same operating frequency measured for FF corner in Table 6.1, but at the same time leakage is 30 times lower since threshold voltage is not modified.

V_{DD}	P_{read_0}	P_{write_0}	P_{read_1}	P_{write_1}	$P_{leakage}$
$1.62\,V$	$0.87\,mW$	$1.87\,mW$	$0.90\,mW$	$5.14\,mW$	$0.75\,mW$
$1.8\,V$	$0.99\,mW$	$2.36\,mW$	$1.06\,mW$	$5.70\,mW$	$1.41\,mW$
$1.98\,V$	$1.22\,mW$	$2.86\,mW$	$1.38\,mW$	$10.12\,mW$	$3.56\,mW$

Table 6.6: Measured leakage and medium power during reading and writing of the 32 x 32 Register File working with 4 different temperature values.

Chapter 7

Conclusions

This work establishes a starting point to introduce the possibility of generating a Register File through an open-source software. The most important feature of a Register File, so the possibility to perform more than one access in the same cycle, leads to enlarge the memory bandwidth and enables the use of those components inside high speed processors at a cost of a reasonable area overhead.
This project provides an updated version of OpenRAM framework that ensures the creation of a multiported memory based on the hardware structure of a Register File with 2 read ports and 1 write port, composed by all the modules whose design is exposed in the previous chapters.
The design is based on a completely open-source flow. The technology used is Sky130, whose PDK is open-source as well, that was already integrated in OpenRAM before the starting point of this work.
A very common configuration of a Register File, 32 x 32, is considered to evaluate performance, area and power features of the designed hardware structure by considering the process corners provided by the SkyWater130 PDK and with different values of temperature and supply voltage. Maximum operating frequency and power results are evaluated thanks to a SPICE simulation, while area information is provided by the memory datasheet provided by OpenRAM after compilation. Timing and power results given by OpenRAM are very approximated since they are derived by using an approximated analytical model. It would be possible to obtain more precise results through

a complete simulation, but it is very burdensome from computational point of view.
This work could inspire further projects about the improving of the designed configuration to obtain better results and the creation of a Register File with this configuration and an higher number of ports by taking advantage of the framework that needs few adjustments to be ready to perform such a task.

Bibliography

[1] Matthew R Guthaus, James E Stine, Samira Ataei, Brian Chen, Bin Wu, and Mehedi Sarwar. OpenRAM: An open-source memory compiler. In *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–6. IEEE, 2016.

[2] David Money Harris Neil H.E. Weste. *CMOS VLSI Design: A Circuits and Systems Perspective, Fourth Edition*. Addison-Wesley, 2010.

[3] Hoan Nguyen, Jihoon Jeong, Francois Atallah, Daniel Yingling, and Keith Bowman. A 7-nm 6R6W Register File With Double-Pumped Read and Write Operations for High-Bandwidth Memory in Machine Learning and CPU Processors. *IEEE Solid-State Circuits Letters*, 1(12):225–228, 2018.

[4] Ram K Krishnamurthy, Atila Alvandpour, Ganesh Balamurugan, Naresh R Shanbhag, K Soumyanath, and Shekhar Y Borkar. A 130-nm 6-GHz 256/spl times/32 bit leakage-tolerant Register File. *IEEE Journal of Solid-State Circuits*, 37(5):624–632, 2002.

[5] Pei-Yao Chang, Tay-Jyi Lin, Jinn-Shyan Wang, and Yen-Hsiang Yu. A 4r/2w Register File Design for UDVS Microprocessors in 65-nm CMOS. *IEEE Transactions on Circuits and Systems II: Express Briefs*, 59(12):908–912, 2012.

[6] SkyWater SKY130's documentation. Avialable online at the link: https://skywater-pdk.readthedocs.io/en/main/.

[7] Tim Ansell. FOSSi Dial-Up Tim Ansell - Skywater PDK: Fully open source manufacturable PDK for a 130nm process, 2020. Avialable online

at the link: https://www.fossi-foundation.org/2020/06/30/skywater-pdk.

[8] Yuichiro Ishii, Yasumasa Tsukamoto, Koji Nii, Hidehiro Fujiwara, Makoto Yabuuchi, Koji Tanaka, Shinji Tanaka, and Yasuhisa Shimazaki. A 28nm 360ps-access-time two-port SRAM with a time-sharing scheme to circumvent read disturbs. In *2012 IEEE International Solid-State Circuits Conference*, pages 236–238. IEEE, 2012.

[9] Y Ishii, H Fujiwara, K Nii, H Chigasaki, O Kuromiya, T Saiki, A Miyanishi, and Y Kihara. A 28-nm dual-port SRAM macro with active bitline equalizing circuitry against write disturb issue. In *2010 Symposium on VLSI Circuits*, pages 99–100. IEEE, 2010.

[10] Bharadwaj S Amrutur and Mark A Horowitz. A replica technique for wordline and sense control in low-power SRAM's. *IEEE Journal of solid-state circuits*, 33(8):1208–1219, 1998.

[11] http://bnrg.eecs.berkeley.edu/ randy/Courses/CS252.S96/Lecture05.pdf.

[12] https://github.com/VLSIDA/OpenRAM.