Tecnológico de Costa Rica

Escuela de Ingeniería Electrónica

Design of a custom SRAM compiler in 180 nm using the Synopsys tool suite

Documento de tesis sometido a consideración para optar por el grado académico de Maestría en Electrónica con Énfasis en Microelectrónica

Felipe Herrero Chavarría

Cartago, 29 de octubre, 2021

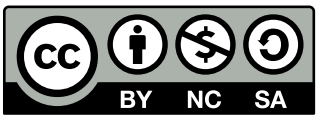
Declaro que el presente documento de tesis ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos y resultados experimentales propios.

En los casos en que he utilizado bibliografía he procedido a indicar las fuentes mediante las respectivas citas bibliográficas. En consecuencia, asumo la responsabilidad total por el trabajo de tesis realizado y por el contenido del presente documento.

Felipe Herrero Chavarría

Cartago, 29 de octubre de 2021

Céd: 1-0123-0456



Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Tesis de Maestría
Tribunal Evaluador

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

Miembros del Tribunal

JOHAN CARVAJAL GODINEZ (FIRMA)
Firmado digitalmente por JOHAN CARVAJAL GODINEZ (FIRMA)
Fecha: 2022.02.10 18:46:31 -06'00'

Dr. Johan Carvajal Godínez
Coordinador de Maestría

Firmado por RONNY GIOVANNI GARCIA RAMIREZ (FIRMA)
PERSONA FISICA, CPF-01-1137-0229.
Fecha declarada: 11/02/2022 08:22 AM
Esta representación visual no es fuente de confianza. Valide siempre la firma.

Dr. Ronny García Ramírez
Profesor Lector

ROBERTO CARLOS MOLINA ROBLES (FIRMA)
Digitally signed by ROBERTO CARLOS MOLINA ROBLES (FIRMA)
Date: 2022.02.11 20:45:41 -06'00'

M.SC. Roberto Molina Robles
Profesor Lector

PABLO DANIEL MENDOZA PONCE (FIRMA)
Firmado digitalmente por PABLO DANIEL MENDOZA PONCE (FIRMA)
Fecha: 2022.02.13 14:45:50 -06'00'

Dr. Pablo Mendoza Ponce
Profesor Lector

ALFONSO CHACON RODRIGUEZ (FIRMA)
PERSONA FISICA, CPF-01-0702-0796.
Fecha declarada: 11/02/2022 01:41:44 PM
Razón: Doy fe
Lugar: Cartago Contacto: alchacon@tec.ac.cr

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que la presente tesis de maestría ha sido aprobada y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

29 de octubre del 2021

Instituto Tecnológico de Costa Rica
Escuela de Ingeniería Electrónica
Tesis de Maestría
Tribunal Evaluador
Acta de Evaluación

Tesis de maestría defendida ante el presente Tribunal Evaluador como requisito para optar por el grado académico de maestría, del Instituto Tecnológico de Costa Rica.

Estudiante: Felipe Herrero Chavarría

Nombre del Proyecto: *Design of a custom SRAM compiler in 180 nm with Synopsys tools*

Miembros del Tribunal

JOHAN CARVAJAL GODINEZ (FIRMA)
Firmado digitalmente por JOHAN CARVAJAL GODINEZ (FIRMA)
Fecha: 2022.02.10 18:47:12 -06'00'

Dr. Johan Carvajal Godínez
Coordinador de Maestría

Firmado por RONNY GIOVANNI GARCIA RAMIREZ (FIRMA)
PERSONA FISICA, CPF-01-1137-0229.
Fecha declarada: 11/02/2022 08:23 AM
Esta representación visual no es fuente de confianza. Valide siempre la firma.

Dr. Ronny García Ramírez
Profesor Lector

ROBERTO CARLOS MOLINA ROBLES (FIRMA)
Digitally signed by ROBERTO CARLOS MOLINA ROBLES (FIRMA)
Date: 2022.02.11 20:46:14 -06'00'

M.SC. Roberto Molina Robles
Profesor Lector

PABLO DANIEL MENDOZA PONCE (FIRMA)
Firmado digitalmente por PABLO DANIEL MENDOZA PONCE (FIRMA)
Fecha: 2022.02.13 14:46:46 -06'00'

Dr. Pablo Mendoza Ponce
Profesor Lector

ALFONSO CHACON RODRIGUEZ (FIRMA)
PERSONA FISICA, CPF-01-0702-0796.
Fecha declarada: 11/02/2022 03:29:07 PM
Razón: Doy fe
Lugar: Cartago Contacto: alchacon@tec.ac.cr

Dr. Alfonso Chacón Rodríguez
Profesor Asesor

Los miembros de este Tribunal dan fe de que la presente tesis de maestría ha sido aprobada y cumple con las normas establecidas por la Escuela de Ingeniería Electrónica.

Nota final de la Tesis de Maestría: 100

29 de octubre del 2021

Resumen

La constante reducción de escala de las tecnologías ha llevado a que los sistemas electrónicos sean cada vez más rápidos. Como parte integral de la mayoría de los microprocesadores, las memorias también han seguido esta tendencia. La reducción del área consumida manteniendo alta densidad y baja potencia es el objetivo de cualquier diseñador de memoria. Las memorias estáticas de acceso aleatorio (SRAM) son esenciales en el diseño digital por su capacidad para retener datos durante largos períodos mientras la fuente de voltaje está encendida, proporcionando también velocidades más altas en comparación con otros tipos de memoria. Este documentado presenta el desarrollo de una SRAM IP semi-personalizada, con la correspondiente integración a un flujo automático de síntesis digital. También se incluye el proceso para diseñar y caracterizar una biblioteca personalizada, por medio de un flujo de caja negra para su uso en síntesis física con otras bibliotecas.

Palabras clave: SRAM, Synopsys, diseño personalizado, caja negra, síntesis física, síntesis digital, librerias personalizadas, trazado.

Abstract

The constant downscaling of technologies has led to electronic systems to be each time faster. As an integral part of most microprocessors, memories have also been following this trend. The reduction of the consumed area while maintaining high density and low power is the goal of any memory designer. Static Random Access Memories (SRAM) are mandatory in digital design for their capacity to retain data for long periods while the voltage source is on, providing also with higher speeds compared to other memory types. This document presents the development of a semi-custom SRAM IP, with the corresponding integration to a digital automatic flow using Synopsys tools. The process to design and characterize a custom library is also included, with a custom black box flow for its use in physical synthesis with other libraries.

Keywords: SRAM, Synopsys, custom design, black box, physical synthesis, digital synthesis, custom libraries, layout.

A mi mamá, a Cindy y a Fede. Mis pilares, mis objetivos y mi motivación.

Agradecimientos

Este trabajo es el resultado de la colaboración de entre múltiples estudiantes a lo largo de más de 5 años de investigación. En específico le agradezco a Bernardo Rodriguez, John Junier y Brandon Varela por sus contribuciones a los diseños realizados de diferentes componentes para la SRAM final. También le doy un agradecimiento especial al profesor Alfonso Chacón por todas las enseñanzas a traves de estos años y la oportunidad de formar parte del DCILAB durante este proceso de maestría.

Felipe Herrero Chavarría

Cartago, 11 de febrero de 2022

Contents

List of Figures

Table Index

Chapter 1

Introduction

The constant downscaling of technologies has led to electronic systems to be each time faster. This implies a higher power consumption to maintain the speed and density required. As an integral part of most microprocessors, memories have also been following this trend. The reduction of the consumed area while maintaining high density and low power is the goal of any memory designer. Static Random Access Memories (SRAM) are mandatory in digital design for their capacity to retain data for long periods while the voltage source is on, providing also with higher speeds compared to other memory types. SRAM blocks can be used as cache memories [8], FIFO register banks [9], instruction banks, as look-up tables for complex mathematical functions [10], video and image processing [11] or in applications that require long time storage.

An SRAM module can be characterized by four general parameters: area, power consumption, transaction speed, and reliability. The market and the industry are interested in electronic solutions with high memory storage, low power consumption, and high speeds. When making an SRAM design the designers need to balance these elements since they typically go against one and another. High clock speeds increase the dynamic power consumption. A large number of cells in a memory array will increase static power consumption and access times [12]. Almost any type of improvement on one aspect of an SRAM will imply a trade-off with the rest of the features. These reasons highlight the importance of detailed research on different techniques and architectures on SRAM cells and models. VLSI design teams tend to buy IP blocks for SRAMs instead of making custom memories. Factories already have optimized designs for the manufacturing process and offer compilers to quickly generate reliable designs. This reliability comes at a high economic cost. Also, design teams value the speed of the process to focus on other core aspects of the project. It is possible to obtain a better performance of a chip by designing the memory to satisfy the specific characteristics according to the needs of the project. Since SRAMs also tend to cover a high percentage of the chip area, they contribute heavily to the power consumption of a chip. Another critical aspect of memory design is that the execution of the processor tends to be faster than the access times of the memory, making these blocks the bottleneck for the system.

These reasons highlight the importance of having a fast and reliable design flow for generating complete custom SRAM designs with the specified characteristics for any given project. Having this in mind, the process of designing an SRAM is a highly time-consuming task. To simplify the generation process of custom SRAM designs and not having to depend on a third-party IP, a memory compiler is needed. The memory compiler should consider area, power, and speed parameters for making the SRAM array. The final design also needs to be reliable, given the specific project parameters. Using a memory compiler should be a simple process that lets a design team build SRAMs quickly without needing extensive SRAM knowledge. Thorough testing of the constructed SRAM is also required since memory array distributions can greatly affect the overall performance.

For the past three years, multiple students of the Tecnológico de Costa Rica (TEC), as part of the DCILab, have been working on the design of the custom cells, arrays, and basic periphery circuits for an SRAM. These circuits form the necessary components to create a custom library for synthesis tools. The design of a characterization methodology is required to continue the design process of the SRAM. The main interest in this project for the members of the DCILab is to include a custom SRAM design on a microprocessor for medical applications. Additionally, the DCILab will benefit economically since buying IP for SRAMs is expensive.

1.1 Objectives and document structure

The purpose of this project is to develop an SRAM compiler using Synopsys tools. The workflow has been designed and implemented for the characterization of custom cells. The physical library required to use the SRAM module as a black box was created. Although an array was not generated, the main characteristics for the complete SRAM block were obtained. The methodology for a mixed-signal test environment capable of evaluating standard SRAM test patterns was designed and implemented to assure reliability. As a final result, a 4096 bits memory was evaluated for its inclusion in a microprocessor.

This work is divided into six different chapters, including the first introduction chapter. Chapter 2 presents the state of the art for the basic components of an SRAM memory. Chapter 3 summarizes the primitive physical blocks that form the SRAM design presented in this document. In chapter 4 the proposal of the SRAM compiler flow and environment settings are explored. This includes the creation of custom logic and physical libraries, the creation of custom black box libraries, and the synthesis of custom black box designs with standard libraries. Chapter 5 covers the main simulations done to test the final SRAM design, analyzing the performance of the memory. Finally, chapter 6 gives the conclusions for this investigation.

Chapter 2

State of the Art

This chapter covers the necessary concepts that require the design of a custom SRAM memory array. It is essential to understand the basic functions of the SRAM memory cells, the SRAM functions, required periphery circuits, and array configurations. These provide the descision making foundations during the multiple design steps of the presented memory array and future modifications.

2.1 SRAM Cell

SRAM cells are based on two cross-coupled inverters and two access gates, forming a bi-stable latched circuit. The fundamental characteristic of this type of memory is that it can keep data for an indefinite amount of time, as long as the correct voltage source is applied. This means that the data stored in the cells do not need to be refreshed, unlike DRAM cells [13].

Following the CMOS technology trends, as the years pass there is an increasing demand for high speed, low power, and low area memories. SRAM designs are more compact than flip-flops and are faster than DRAM cells. Compared to a DRAM design, the downside to the SRAM implementation is the lost area required for the inclusion of peripheral circuits that enable the correct access to the memory cells and the power consumption during each access. The cell density that SRAM circuits can achieve makes the increase in the periphery area an overall positive trade-off [1]. Figure 2.1 shows the basic topology for an SRAM cell using 6 transistors.

Both of the outputs are connected to the bit lines (BL) via access n-mos transistors. Write operations are made by driving the desired values on the bit lines while the access transistors are ON with the word line (WL) signal. The read process begins by having the BL precharged to VDD. When the WL has a logic value of 1, the BL discharges through the pull-down net of the inverter. The data can be read from the BL with a Sense Amplifier. Failure in the read and hold operation occurs when an increase in voltage in the internal nodes of the cell passes the inverters tripping point [14].

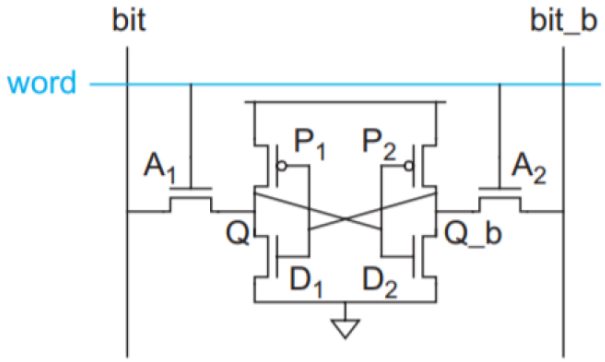


Figure 2.1: 6T basic SRAM cell [1].

2.2 SRAM Array

Multiple individual cells can share the same WL signal to generate a word. These word structures can be stacked by sharing BL between words, forming the memory array. A row decoder is used to select the word that will be accessed. Additionally, three column circuits are required. These circuits are a bitline conditioning circuit, a sense amplifier, and a write driver. Figure 2.2 shows the basic block arrangement for an SRAM memory.

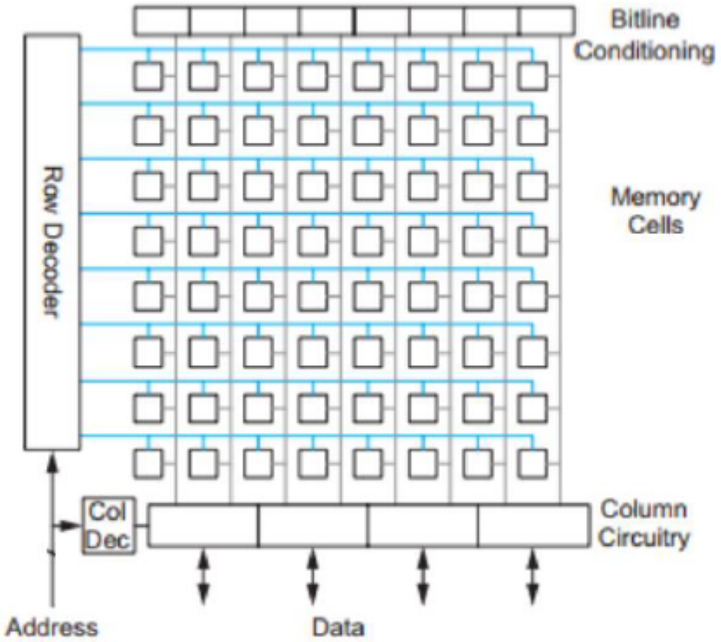


Figure 2.2: Typical SRAM array [1].

The number of words in the memory array is limited by the bitline length. The main component of delay present in the bitline transitions is the capacitance of all the access transistors connected to it. The calculation and impact of this effect is presented in [2].

$$V_b = V_c \times e^{\frac{t}{\tau_c}} \tag{2.1}$$

$$\tau_c = (R_c + R_a) \times (nC_a + C_p) \tag{2.2}$$

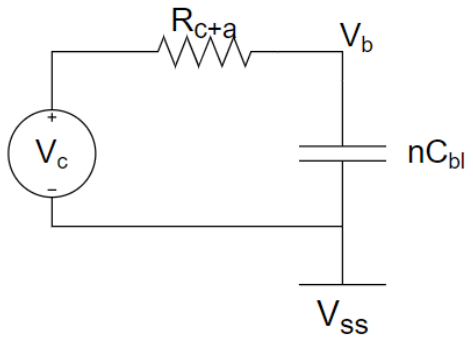


Figure 2.3: Bitline RC model. Adapted from [2]

As seen in Figure 2.3, the bitline can be simplified into an RC circuit. The voltage of the bitline follows the characteristic equation of a capacitor (2.1). The time constant (τ) in (2.2) includes the resistance of the pull-down transistor (R_c), the resistance of the access transistor (R_a), the number of access transistors (n), the capacitance an individual access transistor (C_a) and the capacitance of the input node to the sense amplifier (C_p).

A column decoder can be implemented to add more words to an array without increasing the negative effects of long bitlines. The column decoder selects a particular column in the memory array for reading the contents of the selected memory cell or to modify its contents. The column selector is based on the same principles as those of the row decoder. The major modification is that the data flows both ways, that is either from the memory cell to the output signals (read cycle), or from the input signals to the cell (write cycle) [15].

2.3 Write Driver

The four steps to make a correct write process: precharge, row and column selection, force BL value, and the change of the value in the cell nodes. Writing data into an SRAM cell requires the inversion of the stable state on a cell. Write drivers set a strong logic 0 on the desired BL and leave the other BL with a weak logic 1. After the value is set, the WL of a specific row is selected. The node with the driven bit line discharges first through the write driver. This also activates the pull-up transistor of the opposing node, setting a strong logic 1. Figure 2.4 shows in a graphical way the process of a writing cycle and Figure 2.5 shows a basic diagram of the write driver. The work presented in [16] explains with greater detail the four stages of writing data into SRAM cells.

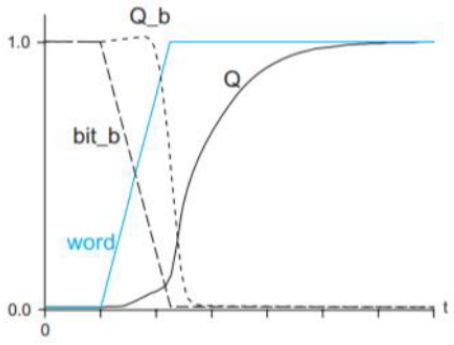


Figure 2.4: Write process of an SRAM cell [1].

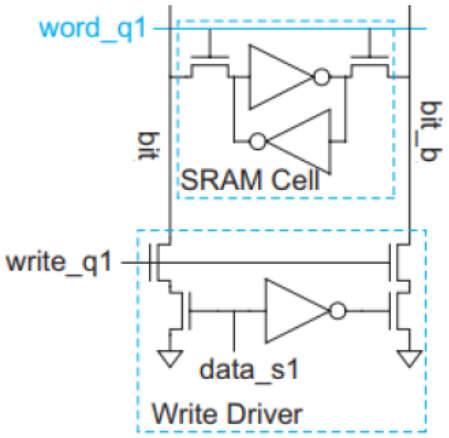


Figure 2.5: Basic write driver [1].

2.4 Sense Amplifier

The reading process of an SRAM starts by having both BL floating after a precharge cycle. After the WL is selected, one of the BL discharges through the access transistor and pull-down transistor of the node storing the logic 0. This process makes electric charges pass through the inner node of the cell and should not invert the cell during this process. Once the BL is discharged, the BL value is detected and data is read [1]. The dynamic energy consumption of an SRAM for a read operation, shown in [16], is given by:

$$E_R = C_P(V_{DD})^2 - \frac{1}{2}(V_{BL})^2 C_{BL} + (2^N - 1)\left[C_{BL}(V_{DD})^2 + \frac{1}{2}\frac{(I_{PT}\Delta T - C_{BL}V_{DD})^2}{C_{BL}}\right] \quad (2.3)$$

Fully discharging the BL has a high power and delay cost. From (2.3), the main ways to reduces energy consumption during read cycles (without reducing V_{DD}) is to reduce the BL voltage as little as possible and make the read cycle as fast as possible. In order to achieve these two points, sense amplifiers can be implemented. The purpose of any type of sense amplifier is to replicate the inner values of a data cell by detecting a small voltage difference between two BL. As seen in (2.1), a small difference in the BL voltage

is achieved by accessing the cell for a small amount of time. This process also reduce the power consumption during the next precharge cycle. Figure 2.6 shows a latched sense amplifier.

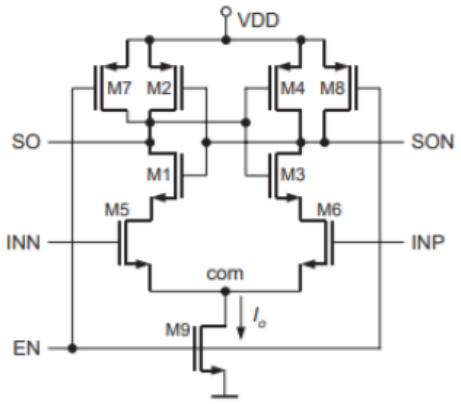


Figure 2.6: Latched sense amplifier [3].

2.5 Row Decoder

SRAM row decoders, or address decoders, propagate the control signals to select a single WL that are to be accessed during read and write cycles. During the precharge phase, the control circuit blocks the decoder signal to ensure none of the words are active. SRAM row decoders can be divided into three sub-blocks: row latches, decoder logic, and row buffers.

The first step for the address decoders is the capture of the input signals. Securing a steady input address before propagating the signals reduces the number of glitches. To capture the signal a latch is used. Once a control signal is set on, the latch becomes transparent between the input and the output nodes [1]. At this point, the address is being decoded by the combinational decoder logic. The decoder logic takes an input address and activates a single output line. Since decoders tend to have a high logic effort and each logic step contains high fanouts, precoding circuits can be implemented. The precoding circuits consist of NAND and NOR trees to reduce logic effort [17].

In larger sized memories, WL signals can have a high capacitance value associated due to the number of access transistors connected for each memory cell. Additionally, WL signals require steep slopes to help the functions of sense amplifier which, as seen on (2.3), require fast access and a minimal change on the BL node voltage. For this purpose, a row driver is included for each output of the decoder [17].

2.6 Control Circuit

Control circuits simplify the implementation of an SRAM and create a general interface for most SoC. The controller circuits take care of the signal synchronization, which can determine the operating characteristics of the memory. For that reason, each controller needs to be adjusted to the specific SRAM architecture implemented. The essential signal for any SRAM controller should be: input address, input data, read signal, write signal, and output data.

SRAM controllers can define the type of operation the memory is going to present. Controllers for SRAMs can be synchronous or asynchronous. Synchronous controllers use clock signals in order to manage input and output registers while asynchronous controllers utilize state machines to run through the various operations of the memory. Synchronous controllers are subdivided into pipelined controllers and flow-through controllers. Pipeline controllers utilize separate registers for the input and output of the data while flow-through ones only use registers for input data [18].

Chapter 3

Design of an SRAM primitive physical blocks

The process of making an SRAM requires several steps that contain the design of multiple analog and digital blocks and cells. These blocks include the basic cells, memory arrays, periphery circuits, and control units. All of the custom designs presented in the following sections have been made on Custom Compiler, reaching a post-layout stage using a 180 nm technology. The post-layout designs have also been tested to operate under typical conditions (1.8 V, 25 ℃) and running at 50 MHz.

3.1 SRAM cell and arrays

A 6T cell was chosen as the first approach of design for its reliability and low area. Since the expected applications do not require high speeds, maintaining the area to a minimum was the focus during the schematic and layout creation process. Having this in mind, the access transistors and pull-down net were made larger to increase noise, read and write margins. Figure 3.1 shows the layout implemented for individual SRAM cells and table 3.1 contains the dimensions of the transistors in the cell.

The standard memory array defined in [4] and [6] contains 64 words, each having a size of 32 bits. The array uses mirrored cells, in which each contiguous pair of words share supply or ground lines. In [5] a double array is tested ($64 \times 32 \times 2$). This double array implementation is the first step towards a modular SRAM memory array construction, in

Table 3.1: SRAM basic cell transistor dimensions [4].

Transistor Pair	Width (nm)	Length (nm)
Pull-up	220	270
Pull-down	580	180
Access	290	180

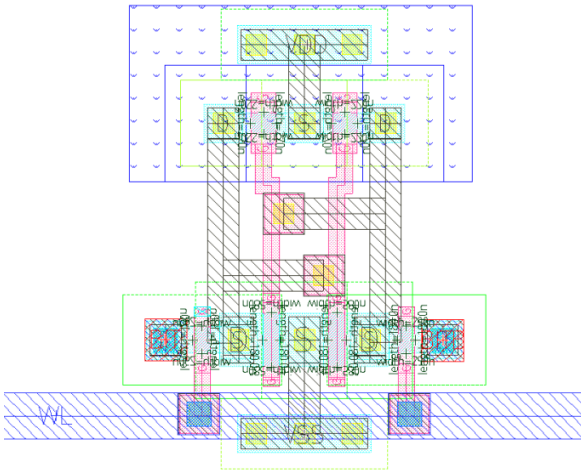


Figure 3.1: SRAM basic cell layout [4].

which two basic arrays can share inputs and outputs without compromising functionality and timing constraints.

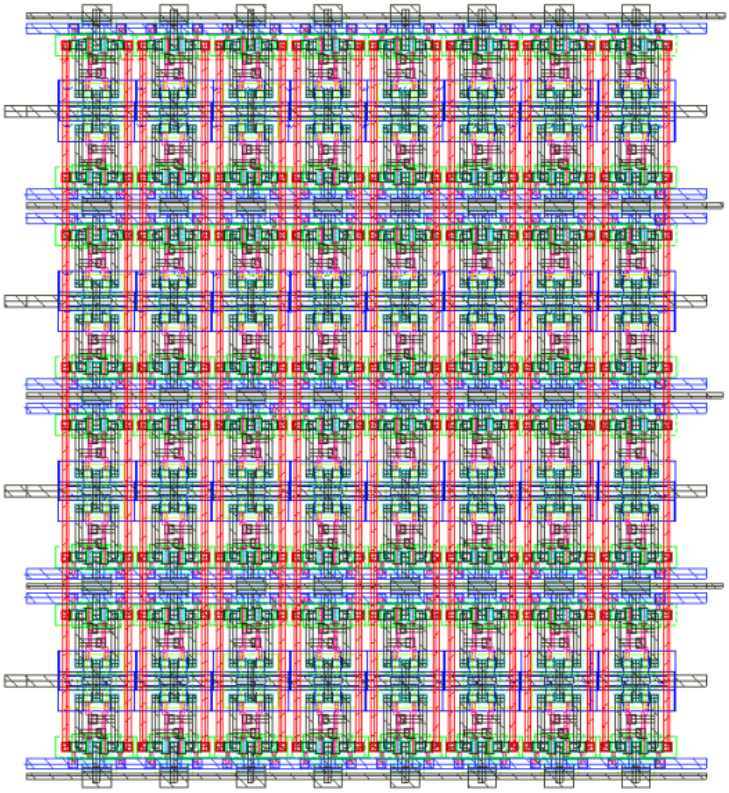


Figure 3.2: SRAM 8 × 8 array [4].

3.2 Periphery circuits

3.2.1 Column peripheral circuits

The SRAM memory array uses three essential column peripheral circuits:

- Sense amplifier [4], [5].
- Write diver [4].
- Precharge circuit [4].

As an optional component, a pass transistor multiplexer [19] is used when implementing modular arrays.

A current sense amplifier is used for this work. The circuit was designed to reduce power consumption during read cycles. The sense amplifiers were tested using a $(64 \times 32 \times 2)$ memory array, reporting read cycles of 2 ns. For the circuit to respond correctly, the word line signal and the sense enable signal need individual control, adding to the complexity of the control circuit. For the write drivers and precharge circuits, the implementations shown in [1] were used at minimum size. Fig 3.3 shows the layout of the used sense amplifier. A drawback for this circuit, not mentioned in [5], is that this type of sense amplifiers have add a layer of complexity to the control unit and the timing of its output signals. Working with small timing windows on custom analog designs can lead to unexpected errors during post layout steps. These errors become more prominent when working on larger and denser designs due to the amount of variability and parasitic components present. In the specific case of sense amplifiers, the main risk are the control signals that do not follow a clock signal edges for proper timing. The read cycles will require additional analog logic to properly generate the access to the bit cells.

The write driver designed and implemented in [4] was the basic design shown in [1]. The dimensions of the write driver took into account that working at minimum size would not provide optimal performance for the SRAM in terms of power consumption and speed, and so a custom size for the driver was proposed for an architecture of 64 words. Throughout the different stages of the design of the SRAM memory, this write driver has not needed any change. The writing times reported have always been lower than read access times, even after the implementation of sense amplifiers. This results should indicate that an even smaller write driver can designed to reduce area and reduce parasitic components added to the data bus during other operations. The Fig. 3.4 shows the write driver used in the presented SRAM design.

The pass transistor multiplexer [19] allows for the use of the periphery circuits between two or more SRAM arrays. This type of multiplexer has a low area cost. In the case of the proposed array, multiple pass transistors in series would degrade the signal value reaching the sense amplifier, compromising the reliability of the read cycles. This means limiting the number of arrays for each periphery group to two. A possible improvement to this design is the study of multiplexing the WL signals instead of the BL signals. The

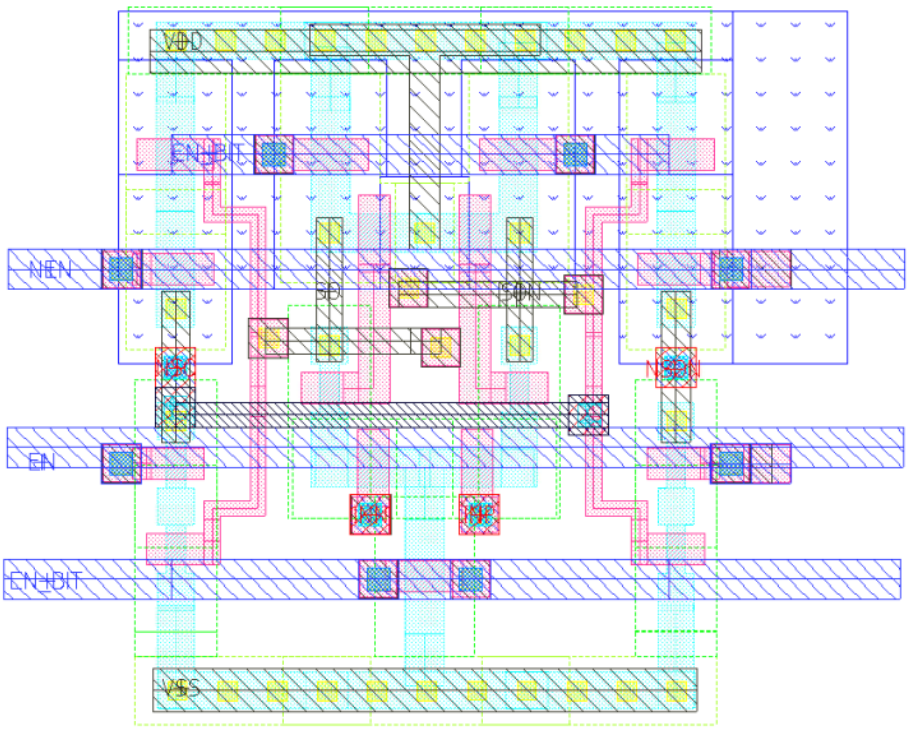


Figure 3.3: Current sense amplifier [5].

multiplexer design would need to be placed between the two SRAM 64 × 32 banks. This would reduce the degradation of the bit line signals during read access but also effectively reducing the power consumption of all accesses since only one bank will be active per read/write cycle.

3.2.2 Decoder

The decoder circuit [6] selects which word line is accessed. The input data of the decoder is propagated throughout the circuit during the pre-charge phase of the memory. When the read or write cycles start, a latch holds the output value and enables the selected word line. This implementation reduces the risk of having false access to cells due to glitches during the propagation of the decoder signals.

In [6] an optimized decoder was made for an 64×32 SRAM. This approach works since one of the basic SRAM arrays to be designed has a size of 64 words. For the cases in which an SRAM needed a different size [20] a modular decoder for SRAM was designed. This decoder is divided into smaller combinational steps that allow the implementation of the desired amount of input and output signals. The basic cells that compose this decoder are made with custom designs that have the same pitch as the SRAM cells and are included in the proposed custom library.

A drawback found throughout the current design is that the load driven by the enable signal should not be driven by a single line. Either a tree structure should be studied to

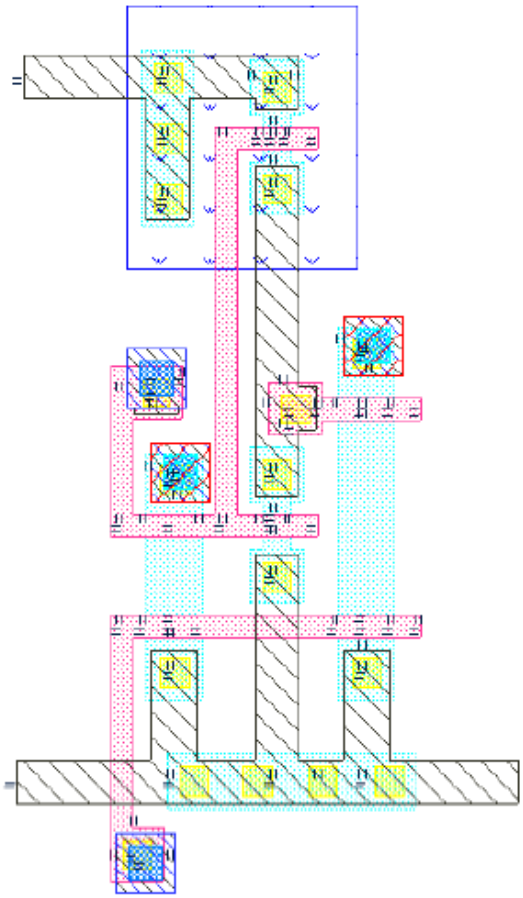


Figure 3.4: Write driver [4].

reduce the load per driver or change the SRAM design to one that uses big signal reading circuits.

In Fig. 3.5 a block diagram for all the previous circuits is presented. Its corresponding layout is given in Fig. 3.6.

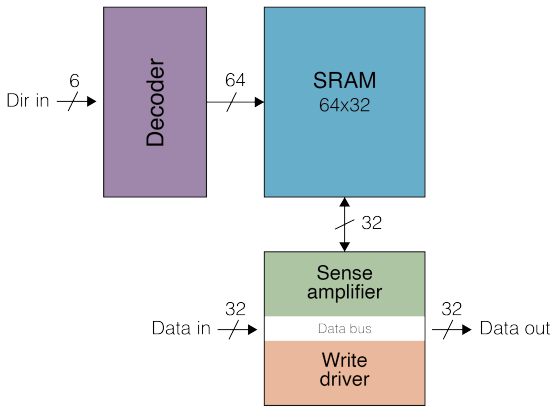


Figure 3.5: Basic SRAM periphery block diagram.

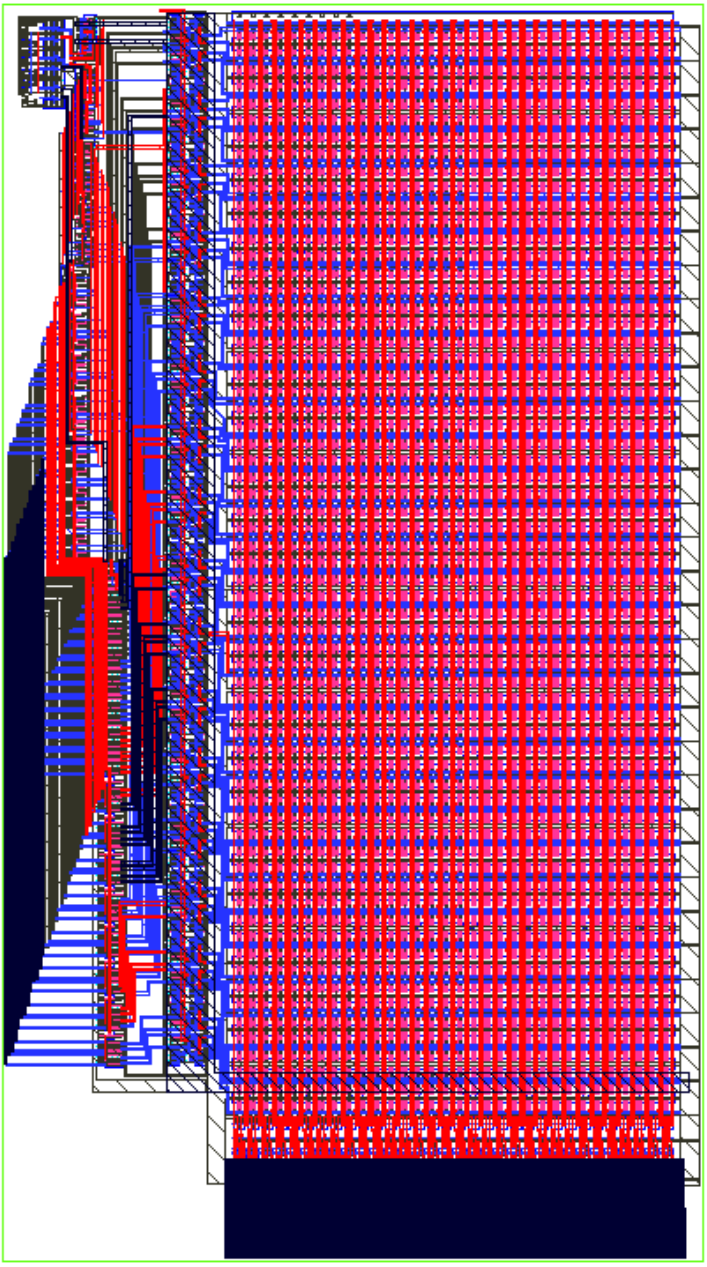


Figure 3.6: 64×32 SRAM layout with periphery circuits and decoder [4], [6].

3.2.3 Controller Unit

The synchronous controller circuit was designed in [21]. The inputs of the SRAM block are reduced to input data, row address, column address, read signal, write signal, clock signal and enable signal. Input signals should be set before the rising edge of the clock cycle, allowing the propagation of the control signals correctly before the read or write phase, starting in the next positive flank. As an additional feature to the controller, a dummy cell implementation was used to increase the accuracy of the timing for the read enable signals.

The cells used in [21] were also made with the custom pitch of the SRAM cells and are part of the proposed custom library. The creation of this custom library and the final layout implementation of this design is covered in the following chapters of this work.

Chapter 4

SRAM compiler

4.1 Workflow and environment

The process to take any custom design from the layout view to a custom library requires multiple environments and tools. A consistent and organized route to follow for this process is not usually found in a manual. The process of making custom SRAMs, and custom libraries in general, is here proposed using the Synopsys workflow.

This workflow should allow the designer to start from any point in the process if the previous steps have been already completed. In the next sections, each of the general steps are described in greater detail. As part of the design for the work environment, a master directory is proposed, containing individual directories for each tool shown in Fig. 4.1.

4.1.1 Layout design and extraction

The tool used for layout design is Custom Compiler. Layouts start from a schematic view, an ideal representation of the components of the design. A layout is created after verifying the correct implementation of a schematic via functional simulations. Layouts contain the physical view of the design. Each layer of a given technology process is contained in this view. A verification process is then applied to check if it follows the design rules of the technology used as a reference. This layout is compared to the schematic to see if both views represent the same circuit in terms of components and interconnections. Lastly, the extraction of the parasitic elements of the design is executed. Post-layout simulations can be applied to the design to obtain an accurate representation of the behavior it is going to present in real applications.

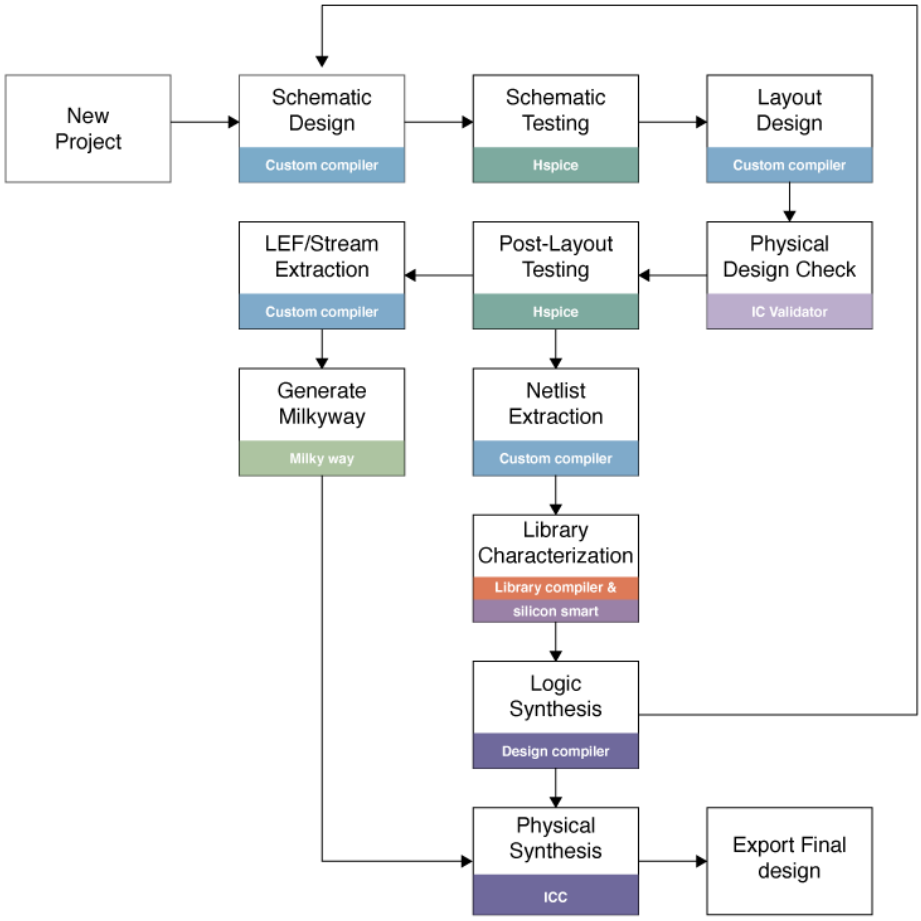


Figure 4.1: General design flow for custom cells

When the post-layout tests are finished Custom Compiler allows to generate three output files required for the next steps:

- Spice file with post-extraction nets.
- Library Exchange Format file.
- Design Exchange Format file.

In case of the SRAM design, the design starts with the individual SRAM cell since this circuit is going to define the pitch for the rest of the elements in the custom library. Some analog circuits such as sense amplifiers can have different pitch sizes if required.

Table 4.1: Layout design and extraction tools and output files.

Required tools	Output Files
Custom Compiler	$cell.spi $cell.LEF $cell.DEF

4.1.2 Library Generation

The library generation process requires several software tools to complete. VLSI libraries contain two mayor groups of information: logic information and physical information. The logic section of the library contains information regarding the cell's timing characteristics, power consumption, functional descriptions, netlist connections, and, as an optional value, area. Physical information on libraries contains layout representations that allow the tools place and route the cells and generate the files required to sent the final design to a foundry.

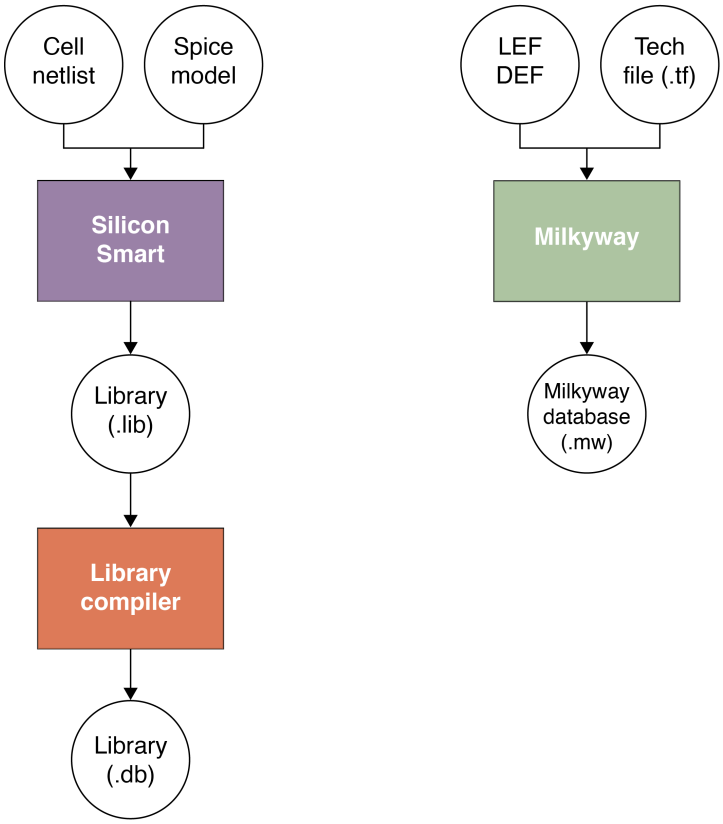


Figure 4.2: Library generation work diagram.

Following the diagram in Fig. 4.2, the left side describes the process for the generation of the logic information section of the library. The right side is the physical information. The cell netlist, LEF, and DEF files are generated in Custom Compiler after completing the layout extraction process. The SPICE model contains the BISM or related model parameters for the components present in the netlist, such as transistors and parasitic elements. Finally, the technology file contains physical design rules and electrical descriptions of the materials used in the physical construction process.

Silicon Smart allows for the characterization of cells. This characterization process builds lookup tables describing the timing and power using multiple output loads and input slopes. Silicon Smart automatically generates test benches that cover all the functions of a given cell to accurately generate the results. The slope and load points are given

by the user. In case a custom library is being made to work alongside other standard libraries, the specified operating points should coincide to maintain consistency in the characterization results. Silicon Smart can also generate other convenient files such as Verilog instances, Verilog descriptions, and datasheets for the cells. The library file made by Silicon Smart contains the logic information of the cells. Lastly Library Compiler takes as an input the .lib file and converts it into the binary format used in other Synopsys tools (.db).

For the physical library, Milkyway takes the LEF and GDS files to generate multiple views of cells. Technology files are used to characterize physical interconnections in the cells and to verify that the generated views still follow the design rules. The Milkyway database should contain the following views:

- CELL view: Complete layout of a structure.
- FRAM view: Abstract representation used for place and route.
- ILM view: Interface logic modules used in optimizations for place and route.
- FILL view: Metal fillings used in the final steps for closing the design and sending to the foundry.

Table 4.2: Library generation tools and output files.

Required tools	Output Files
Silicon Smart	$library.db
Library Compiler	
Milkyway	$library.mw

4.1.3 Design synthesize

When the custom library is generated it can be included into an RTL project and follow a typical VLSI synthesis flow, from the front-end to the back-end (logical and physical synthesis respectively).

The front-end flow is managed by Design Compiler. The input RTL can be written using the Verilog definitions provided by Silicon Smart and setting as target library the custom library previously characterized. Before performing the logic synthesis, a switching activity format file (SAIF) may be generated. To obtain a good SAIF a robust testbench is required. Silicon Smart can also provide the bases for these robust tests, since the test benches used for characterization are saved when the Verilog modules are generated. Design Compiler uses the custom library cells to instantiate a circuit with an equivalent function as the circuit described in the RTL. A preliminary report of timing, power, and area is generated at this point.

The back-end consists of the physical synthesis and verification of timing constraints. The tool used for this type of synthesis is IC Compiler. IC Compiler uses as input the logic synthesis netlist generated by Design Compiler and a new SAIF created out of the

post logic synthesis. IC Compiler also uses the information in the Milkyway database to instantiate the physical cells present on the netlist. Place and route and the inclusion of clock trees and supply grids are also made by IC Compiler. When the synthesis is finished, PrimeTime is used to verify that the final design is correct given the timing restrictions of the project. If the physical specifications are met, the final GDSII file can be generated and exported to other projects or foundries.

Table 4.3: Synthesize design tools and output files.

Required tools	Output Files
Design Compiler Library Compiler	$top_design.GDSII

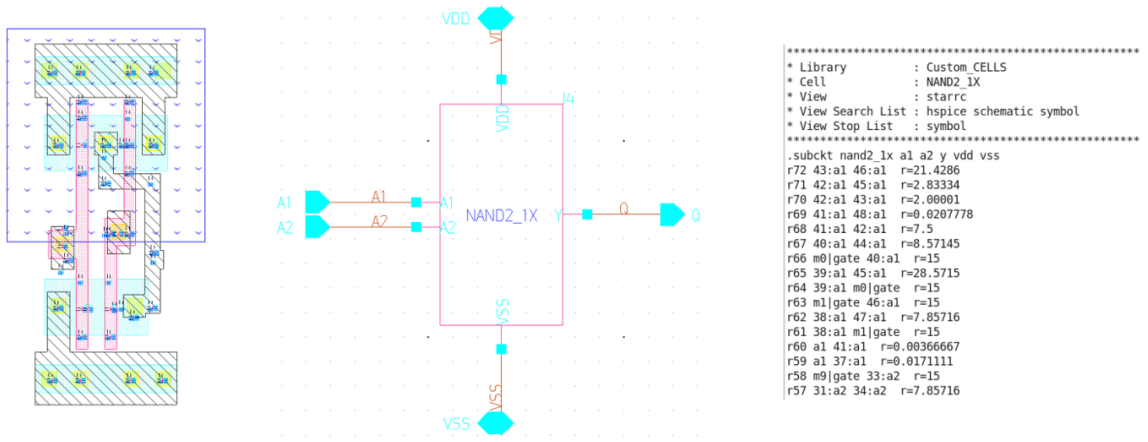
4.2 Cell layout and netlist extraction

When a custom library is needed, it should contain the cells required for a given project. Since the designers at a custom level know exactly which cells are going to be used it is easier to start by listing all the components needed. In the specific case of the SRAM, the core block, control module, and the periphery circuits need to be taken into account. In table 4.4 a list of individual cells needed to build an SRAM is presented. These cells will constitute the custom library. As mentioned in Chapter 3, all of the cells have been tested at a post-layout level. Also, post-layout tests of a $64 \times 32 \times 2$ SRAM were made using control and periphery circuits.

For the extraction of the SPICE netlist in Custom Compiler the STARRC (layout with parasitic extraction) view and symbol view of the cells have to be available. The first step is creating a *config* view and assigning the STARRC view of the desired cell. Although multiple cells can be extracted at the same time it is recommended to extract one cell at a time. This allows the design team to keep order in the work environment and allows for easier access to the individual SPICE files on later portions of the design or in other projects. The SPICE files should be saved on a dedicated netlist directory where Silicon Smart will be used.

Table 4.4: Cell list for proposed custom library.

Required Cells		
AND2_0X [21]	INV_2X [20]	NOR2_1X [20]
BUF_0X [21]	MUX2_0X [21]	NAND2_2X [20]
BUF_1X [21]	MUX4_0X [21]	LATCH_0X [20]
BUF_2X [21]	NAND2_0X [20]	INV_4X [20]
DFFRNQN_0X [21]	TRIBUF_0X [21]	NAND2_1X [20]
DFFRNQN_1X [21]	INV_0X [20]	-
INV_1X [20]	NOR2_0X [20]	-




```
**************************************************
* Library          : Custom_CELLS
* Cell             : NAND2_1X
* View             : starrc
* View Search List : hspice schematic symbol
* View Stop List   : symbol
**************************************************
.subckt nand2_1x a1 a2 y vdd vss
r72 43:a1 46:a1  r=21.4286
r71 42:a1 45:a1  r=2.83334
r70 42:a1 43:a1  r=2.00001
r69 41:a1 48:a1  r=0.0207778
r68 41:a1 42:a1  r=7.5
r67 40:a1 44:a1  r=8.57145
r66 m0|gate 40:a1  r=15
r65 39:a1 45:a1  r=28.5715
r64 39:a1 m0|gate  r=15
r63 m1|gate 46:a1  r=15
r62 38:a1 47:a1  r=7.85716
r61 38:a1 m1|gate  r=15
r60 a1 41:a1  r=0.00366667
r59 a1 37:a1  r=0.0171111
r58 m9|gate 33:a2  r=15
r57 31:a2 34:a2  r=7.85716
```

Figure 4.3: (A) STARRC view of NAND2_1X (B) Config view of NAND2_1X (C) Portion of spice netlist of NAND2_1X

The layout data extraction is made by generating the LEF and GDSII files. These two files should contain all the physical information required of a given cell. The layout must have its boundaries defined, have the correct pin notation for input and output nodes, and have the keep-out zones where required. Custom Compiler can generate both views using the extraction options on the main console. LEF files also need to be extracted with the technology information.

4.3 Custom Library creation

The characterization process of the cells and the compilation of the cell results is made using Silicon Smart. Silicon Smart creates its own directory tree when set up.

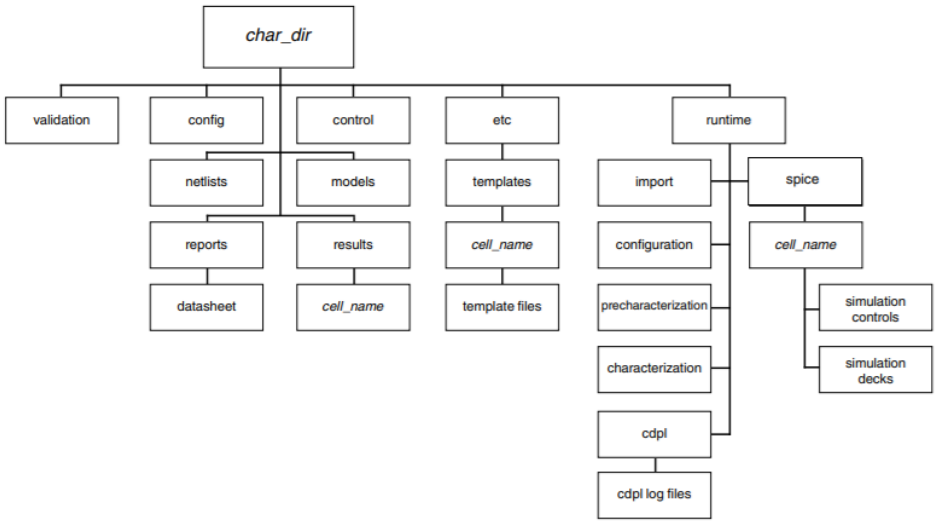


Figure 4.4: Directory tree used by Silicon Smart [7].

The folder where the netlists are saved is copied into the netlist folder of Silicon Smart. A configuration file is also required. This file contains the information necessary to build the characterization test benches and setting up the simulator used to apply the tests. The PVT points of the library can also be modified within the configuration file. The most important part of the configuration file is the definition of the input slopes and output loads. Silicon Smart creates a reference table with all the possible combinations of the selected slopes and loads. Synthesis and routing tools use the reference tables and interpolate the data to find an approximation of the response of a given cell when implemented on a circuit. The cell's characterization will be more accurate if the slopes and loads are within the ranges that the cell is expected to operate. To maintain consistency between the custom library and the standard libraries available, the characterization points are selected to resemble the ones used in the library D_CELLS_HDLL_LPMOS_typ_1_80V_25C, from XFAB. The library containing the model descriptions for the components in the netlist also needs to be added to the configuration file.

The Silicon Smart flow should execute the following characterization stages in order:

- Read configuration file (.tcl).
- Create directory tree.
- Import cells (.sp).
- Recognize cell functions.
- Create characterization models.
- Characterization.
- Generate output files.

The recognize cell function creates instance files of cells that contain the truth table representing the functional behavior of the cell. The instance file also contains the direction of all the ports of the cell. This recognition process is carried out automatically by Silicon Smart by analyzing the netlist and applying a small test to detect the function. To make the process more reliable, Silicon Smart allows us to directly name the different ports and the direction they have. Indicating the clock signals and supply nodes improves the chances of a cell to be recognized. In the case of analog circuits, the instance file needs to be manually made. Instance files can be added to the control directory.

The characterization model is made using the instance files. The testbench stimulates the cells with input vectors that cover all the possible states and transitions. Each group of vectors is applied to each combination of input slopes and output loads set on the configuration file. The test benches and vectors used are stored in Verilog files inside the model's directory. After the tests are ready, Silicon Smart uses HSPICE (or another simulator of choice) to execute the test benches and save the results. The characterization process also includes the compilation of these results into the standard Liberty format (*.lib*).

Finally, the output files are copied in the model's directory. If necessary, Verilog instances of the cells can be made, including their functional description. Another useful file to generate is the library datasheet. The datasheet contains a practical view of the

characterized cells, presenting the truth table, timing characteristics, dynamic power consumption, and static power consumption.

DFFRNQN_0X_mod

Company Name Here (set company_name parm)

tm_op_cond/1.800000/25.0

Wed Aug 12 19:32:44 2020

Cell Attributes

Attribute	Value
area	65.6594 µm^2

FLOP GROUP

Attribute	Expression
Registers	IQ1 IQN1
Clocked On	clk
Next State	d
Clear	(!rn)

OUTPUT FUNCTIONS

Output Pin	Function
q	IQ1
qn	IQN1

Figure 4.5: Flip flop cell pin recognition.

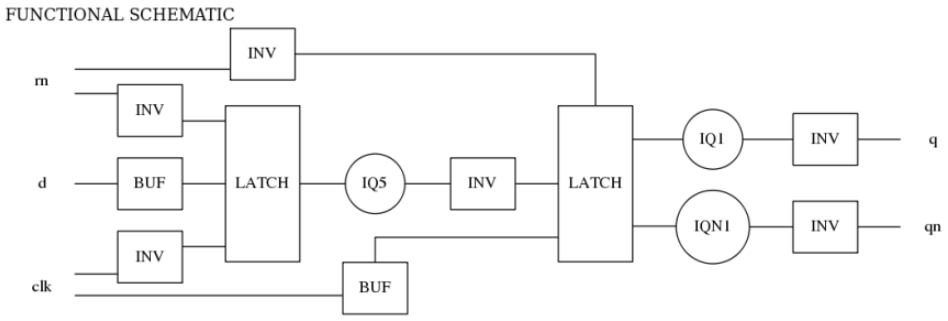


Figure 4.6: Flip flop cell function recognition.

Tables 4.5 and 4.6 contain the parameters used for the Liberty model generation. The input slopes are the same for all the cells. The output loads are specific for each cell. A maximum delay value is assigned. Silicon Smart finds the load capacitance that makes the cell respond with the delay value selected. After that, a linear distribution is calculated between the smallest load capacitance and the value previously found.

Before using Library Compiler, the area of the cells can be added directly to the library in the .lib file. The area of the cells can be measured in Custom Compiler. The units used for the area are set on the configuration file. Library Compiler only requires two commands to convert the custom library (*.lib*) into the binary format used by Synopsys tools (*.db*). Library Compiler needs to read the library file, and if the library has the correct format, the tool then writes the new binary file.

CONSTRAINTS

Constraint Pin	Related Pin	Constraint Pin Tin (ns)	Related Pin Tin (ns)	setup (ns)	hold (ns)
d(LH)	clk(LH)	0.7746	0.7746	0.1419	-0.1183
d(HL)	clk(LH)	0.7746	0.7746	0.3311	-0.2286

Constraint Pin	Related Pin	Constraint Pin Tin (ns)	Related Pin Tin (ns)	recovery (ns)	removal (ns)
rn(LH)	clk(LH)	0.7746	0.7746	0.1813	-0.1656

Constraint Pin	Related Pin	Constraint Pin Tin (ns)	Related Pin Tin (ns)	Minimum Pulse Width (ns)
rn(HLH)	rn(HL)	n/a	n/a	0.4405
clk(HLH)	clk(HL)	n/a	n/a	0.2401
clk(LHL)	clk(LH)	n/a	n/a	0.2114

PIN CAPACITANCE (pf)

Pin	Type	Capacitance (pf)
rn	input	0.0076
d	input	0.0029
clk	input	0.0040

DELAY AND OUTPUT TRANSITION TIME

Input Pin	Output	When Condition	Tin (ns)	Out Load (pf)	Delay (ns)	Tout (ns)
rn(HL)	q(HL)	default	0.7746	0.1212	1.3777	0.7150
rn(HL)	qn(LH)	default	0.7746	0.1207	1.7721	1.2614
clk(LH)	qn(HL)	default	0.7746	0.1207	1.2140	0.7040
clk(LH)	qn(LH)	default	0.7746	0.1207	1.6570	1.2621
clk(LH)	q(HL)	default	0.7746	0.1212	1.3064	0.7380
clk(LH)	q(LH)	default	0.7746	0.1212	1.4695	1.2668

Figure 4.7: Flip flop cell timing constraints.

Table 4.5: Library basic parameters used.

Parameter	Value
Voltage supply	1.8 V
Ground	0 V
Temperature	25 ℃
Current unit	μ A
Leakage power unit	1 pW
Resistance unit	1 kΩ

4.4 Logic synthesis using custom libraries

The logic synthesis is executed by Design Compiler. The tool takes a Verilog file and generates a functionally equivalent circuit using cells from the selected target library. It is also possible directly create instances of the target library cells. The synthesis is also a

Table 4.6: Characterization options used.

Parameter	Value
logic_high_threshold	0.8 %
logic_low_threshold	0.2 %
numsteps_slew	7
numsteps_load	7
smallest_slew	11.4 e-12
largest_slew	6.1158 e-9
smallest_load	1 e-15
max_tout	6.1158 e-9

way to check if the custom library has been characterized successfully. To test the custom library a 6-to-64 decoder has been implemented, using direct instances of the cells.

The first step to make the logic synthesis is to create a switching activity interchange format (SAIF) file. This requires an exhaustive testbench that stimulates all the circuit nodes. Usually, a random input test is implemented and multiple clock cycles are used. In the case of the 6-to-64 decoder, 10,000 cycles where executed with random inputs. Following this, the enable signal is turned off for 1,000 cycles while still randomly changing the input signal. Finally, another 10,000 cycles with the enable signal on. To correctly instance the cells for an RTL simulation, the function.v and function_udp.v files made by Silicon Smart need to be included, as well as the individual Verilog files of the cells.

When the RTL level SAIF is ready, the Design Compiler tool can be executed. Reports for timing, area, and power are set to be made at the end of the synthesis. For the synthesis, only the Verilog files containing the instance of the used cells are used. At this stage, metal orientations can be defined for the construction of a pre-layout mapping. For this test, and every synthesis carried out throughout this project, metals 1, 3, and METTP are set horizontal while metals 2, 4, and METTPL are vertical. The pre-compiled netlist is saved in a *.ddc* file and the constraint file is loaded. After this, the design compilation is made. To make the report files, the SAIF file is used with the compiled design.

```
Library(s) Used:

    tm_op_cond (File: /mnt/vol_NFS_Zener/WD_ESPEC/fherrero/SS/LC_outputs/liberty_tm_op_cond.db)

Number of ports:                          412
Number of nets:                           497
Number of cells:                          244
Number of combinational cells:            206
Number of sequential cells:                 7
Number of macros/black boxes:               0
Number of buf/inv:                         18
Number of references:                      34

Combinational area:               2205.671179
Buf/Inv area:                      119.109598
Noncombinational area:             288.826790
Macro/Black Box area:                0.000000
Net Interconnect area:      undefined  (No wire load specified)

Total cell area:                  2494.497969
Total area:                 undefined
```

Figure 4.8: Area report for 6-to-64 decoder logic synthesis using custom library.

```
  Cell Internal Power  = 145.4199 uW   (35%)
  Net Switching Power  = 272.7259 uW   (65%)
                         ---------
Total Dynamic Power    = 418.4458 uW  (100%)

Cell Leakage Power     =   4.5050 nW

Information: report_power power group summary does not include estimated clock tree power. (PWR-789)

                 Internal         Switching           Leakage            Total
Power Group      Power            Power               Power              Power   (   %    )  Attrs
--------------------------------------------------------------------------------------------------
io_pad             0.0000           0.0000            0.0000             0.0000  (   0.00%)
memory             0.0000           0.0000            0.0000             0.0000  (   0.00%)
black_box          0.0000           0.0000            0.0000             0.0000  (   0.00%)
clock_network      0.0000           0.0000            0.0000             0.0000  (   0.00%)
register           0.0000           0.0000            0.0000             0.0000  (   0.00%)
sequential     6.7539e-02           0.1081          352.0200             0.1756  (  41.99%)
combinational  7.7881e-02           0.1647       4.1530e+03             0.2426  (  58.01%)
--------------------------------------------------------------------------------------------------
Total              0.1454 mW        0.2727 mW     4.5050e+03 pW         0.4182 mW
```

Figure 4.9: Power report for 6-to-64 decoder logic synthesis using custom library.

```
Point                                          Incr       Path
-------------------------------------------------------------
LATCH_0X_inst5/clk (LATCH_0X)                  0.00       0.00 r
LATCH_0X_inst5/q (LATCH_0X)                    1.94       1.94 r
NNORS_0X_inst36/A (NNORS_0X_0)                 0.00       1.94 r
NNORS_0X_inst36/U1/y (INV_0X)                  0.37       2.31 f
NNORS_0X_inst36/U5/y (NOR2_0X)                 0.19       2.50 r
NNORS_0X_inst36/Q[3] (NNORS_0X_0)              0.00       2.50 r
U69/q (AND2_0X)                                0.14       2.63 r
Y[63] (out)                                    0.00       2.63 r
data arrival time                                         2.63
-------------------------------------------------------------
```

Figure 4.10: Timing report for 6-to-64 decoder logic synthesis using custom library.

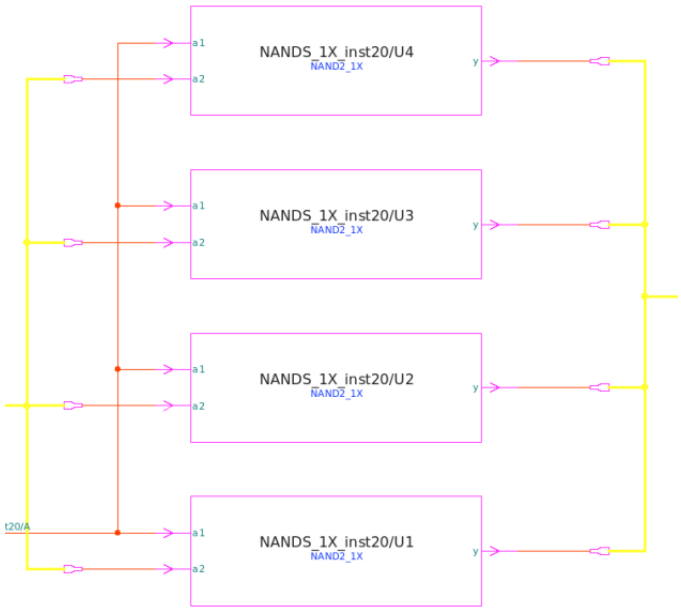


Figure 4.11: Graphical netlist representation showing custom cell instances.

4.5 Physical Library Preparation

In order to use the custom cells in a physical synthesis flow some collateral files need to be generated. In the case of this project, a Milkyway library needs to be created to be used by ICC. The Milkyway library can be made using the tool Milkyway by Synopsys. This tool has several methods to create the Milkyway libraries depending the needs. This section will present the proposed flow used to create the Milkyway libraries of custom standard cells and the SRAM block.

4.5.1 Abstract View setup and creation

The first step to prepare the cells in a Milkyway library is to create an abstract view of the layout. Abstract views mark the physical geometry of pins and remove all the information related to nets and interconnections. The only element maintained from the schematic information is the input and output ports name and directions and their physical position.

Abstract views also contain place and route (PR) boundaries and metal blockages. These boundaries and blockages let PR tools know where metal routes and other cells cannot be placed to ensure DRC rules are not violated. The following subsection will present how to correctly make an abstract view of a layout to use in physical libraries using Custom Compiler.

The process to create an abstract view starts after the layout view has passed DRC and LVS checks. Layouts that belong to the same library need to have the same vertical pitch and horizontal pitch. These two pitch distances make the unit tile of the library. It is recommended to have tested the designs in a post-layout simulation to ensure that the layout used for abstraction is the final design and does not need any more changes. Tools do not check that abstract views and layout views of cells are consistent with each other, making possible that DRC errors appear after physical synthesis if these two views do not represent the same layout. The two elements that Custom Compiler needs to create an abstract of the design are the physical pin information and the PR boundary. Fig. 4.12 shows a layout with physical pins marked and a PR boundary defined.

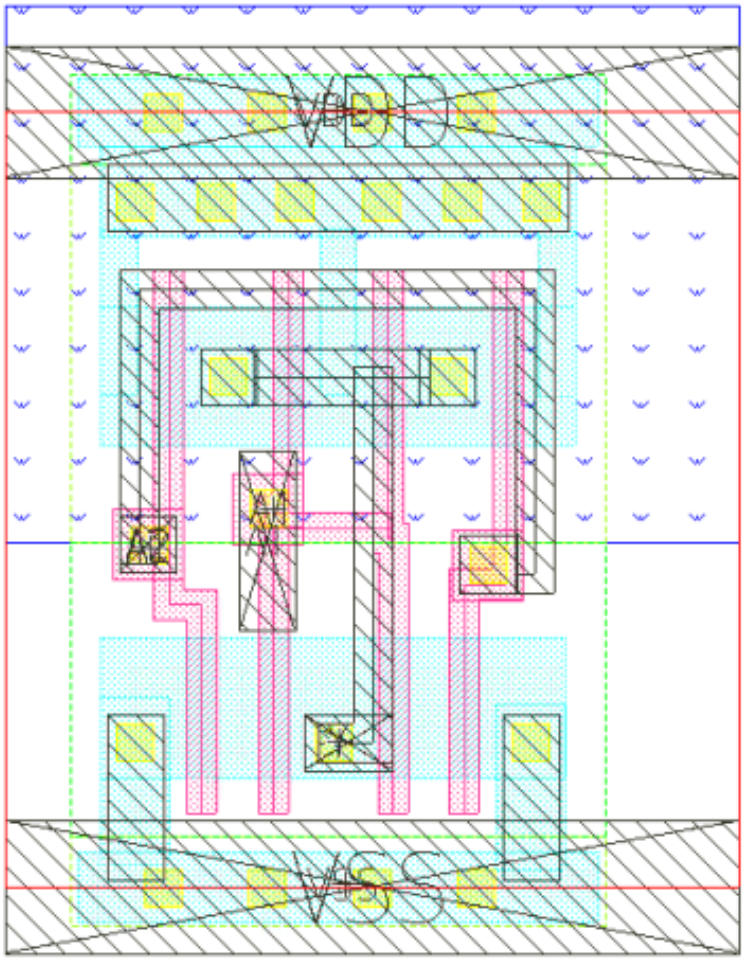


Figure 4.12: NAND layout example with physical pins (nets marked with a cross and pin name) and PR boundary set (red box set around the design).

When setting the physical pins the following considerations need to be taken into account:

- For signal pins correctly select direction (input or output).
- For power pins select the direction as input/output pin.
- Clock ports can be defined as signal pins as well.
- Pins should stay on Metal 1 layer to make the PR process more reliable when using custom cells.
- Pins should have a defined physical area inside the width of at least one unit tile.
- Manually check if a via to Metal 2 can be connected to the pin inside the PR boundary without breaking DRC rules to ensure that the pin can routed.

The PR boundary lets Custom Compiler know the where to create the metal blockage layers. It is important to know that if the library is going to be used in a shared P/G placement process, the PR boundary needs to be set in the middle of the power and ground rails. Shared P/G placement also requires that the contacts between the P/G rails and the diffusion layer are placed in the same position. Contacts are placed in the middle of unit tiles to ensure that they do not cause any DRC issues with other cells with the same unit tile.

After these elements are set, Custom Compiler can create the abstract. When working with standard cells, only the metal layers that are present in the design need to be selected for blockage creation. In hierarchical designs, the level of depth needs taken into account needs to be selected. In Fig. 4.13 the abstract of the layout of Fig. 4.12 is presented.

4.5.2 LEF Extraction

The LEF view or file contains a physical abstraction of the metal layers in a layout. This view contains input and output pin information, as well as PR blockages. The LEF file can be extracted from any layout that has an abstract view. A LEF file can be made for each library. This means that multiple cells can be extracted into a single LEF. It is important to select the abstract view to ensure that only the information necessary for PR is going to be taken into account. As an additional option, LEF files are created to include technology information. The technology information contains some physical rules of the technology process used to make the layouts, such as via information and metal layer pitches.

4.5.3 GDSII Extraction

The GDSII view or stream file contains a complete physical representation of a layout. This view only has the physical geometries of the layers. It does not include any information related to nets. GDSII view can be created from any layout design. In contrast with the LEF extraction process, a GDSII file needs to be made for each cell that is going to be included in the Milkyway Library. After selecting the cell, it is necessary to indicate a

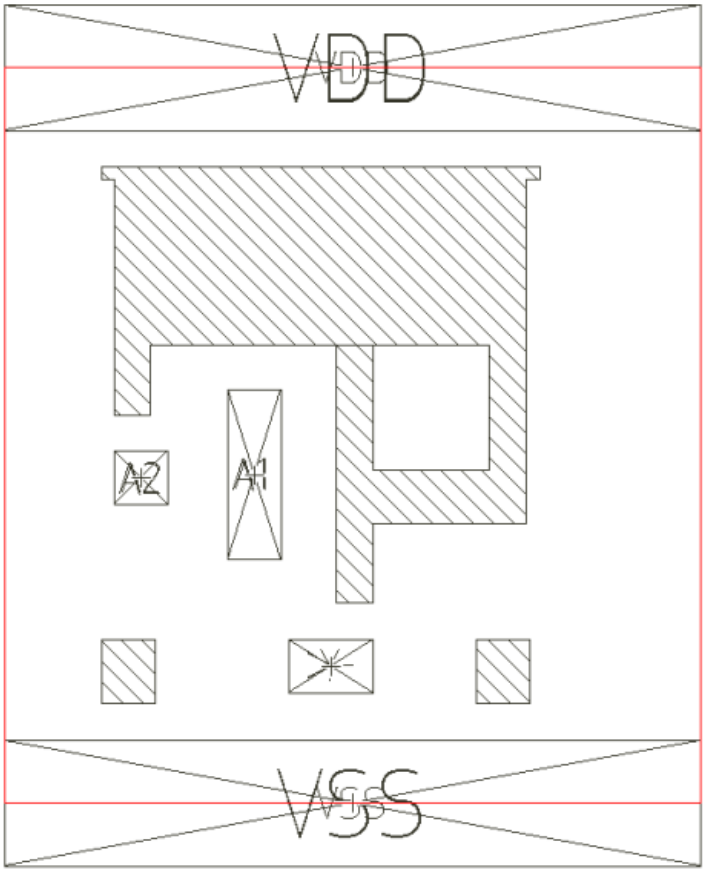


Figure 4.13: NAND abstract view

layer mapping file. This file contains numerical identifications for the process layers. To reduce possible errors, use the same mapping file selected for Custom Compiler (check the Tech Lib box under the Map Files tab).

4.5.4 Milkyway generation flow

The final step to create a physical library is to use the LEF and GDSII files in Milkyway. An script was designed to automatize the use of this tool. The script requires the following:

- Directory with the desired .LEF file.
- Directory with all the desired .gds files.
- A technology file (the same used in Custom Compiler).
- A cell type file.
- A valid unit tile name and dimensions.

Each physical cell in a Milkyway library has two components: a FRAM view and a CEL view. FRAM views contain the abstract representation of a cell that is needed in PR processes. CEL views include a complete layout structure, used as a physical representation for manufacturing. Before running the script in Milkyway, modify the

desired Milkyway name, directory variables, the cell type file, and the unit tile values to correspond to the working project. Each cell should have a cell type so that ICC can process correctly the CEL and FRAM views. TCL expressions can be used to set the types. The available cell types are:

- Hard macro.
- Soft macro.
- Black box.

The script first creates the FRAM view using the LEF files provided for each cell inside the LEF file. A FRAM example is presented in Fig. 4.14. After creating the FRAM, a unit tile is assigned to the Milkyway library. At this point, a CEL view is also present, but it only contains the abstract information. To update the CEL views the script overwrites this file with the information present in each GDSII file. At the same time, the cell type is updated to reflect the values set on the cell type file. Fig. 4.15 shows an example of a CEL view.

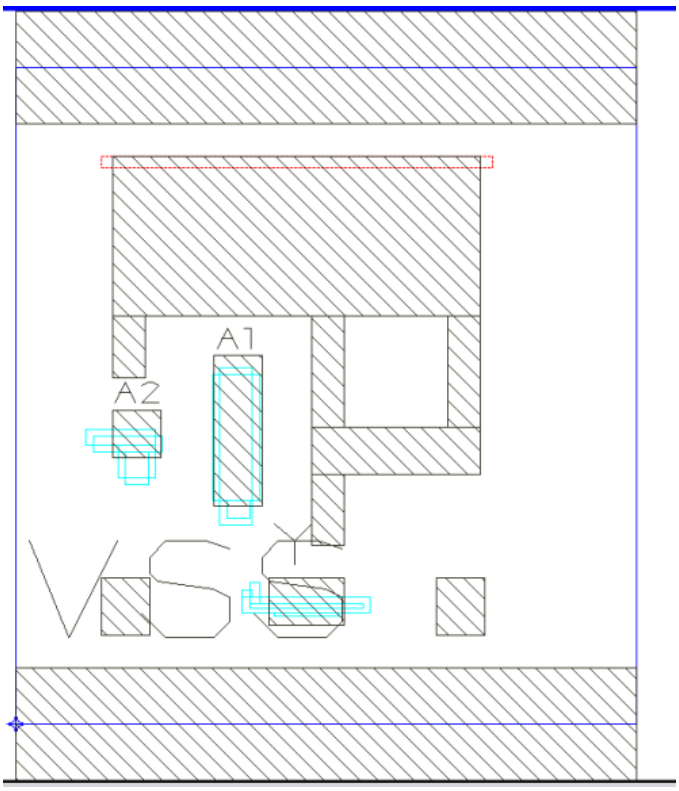


Figure 4.14: NAND FRAM view

4.6 Custom floorplan and place and route

As seen in previous sections, logic synthesis requires that the cells have a Synopsys library file. When cells do not have a standard logic behavior, like analog circuits, Silicon Smart will not be able to identify a Boolean table to represent the cells. Without this description,

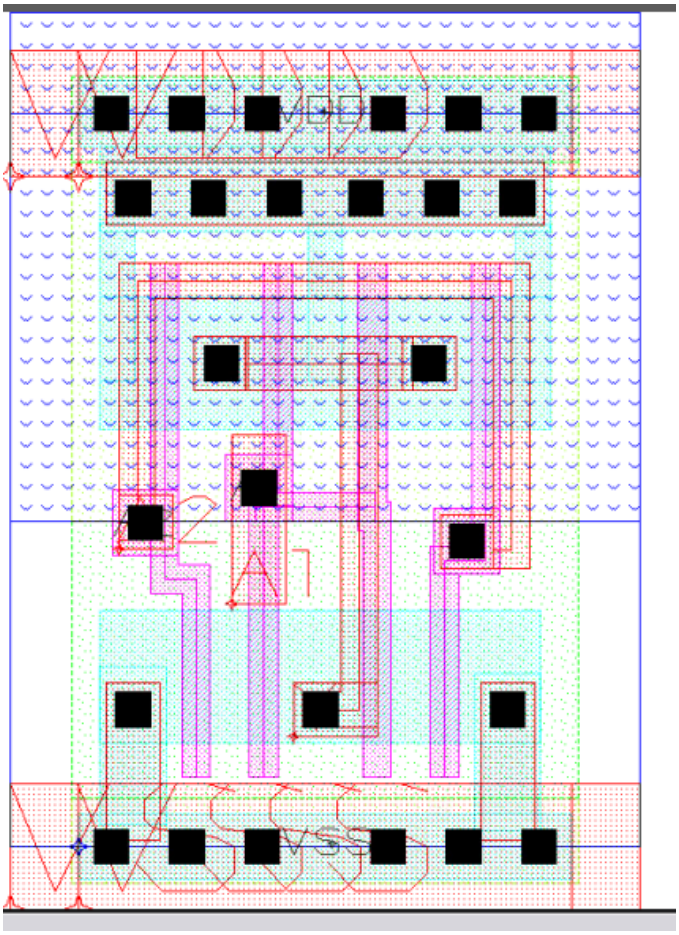


Figure 4.15: NAND CEL view

the tool does not generate valid library information for the cells. Custom Compiler serves as a workaround for this type of situation.

Custom Compiler can read synthesized Verilog files from Design Compiler. A synthesized file will contain the reference names (ref_name) for the custom cells used. Custom Compiler can use this to create an HSPICE netlist. This process can be made automatically using the Design menu in Custom Compiler and selecting the create new cell view from the cell view option. A single library that contains cells that match all the reference names needs to be selected. Figures 4.16 and 4.17 shows an example of a synthesized Verilog module with a custom logic library and its automatically generated netlist.

Having a Custom Compiler netlist view of a circuit allows us to make analog simulations over synthesized circuits. More importantly, a layout can be manually made for the circuit. The challenging side to this approach is that making a custom layout for a big synthesized circuit can be tedious and time-consuming. The construction process of such a design can also lead to human-induced errors. For this type of work, a different approach to custom layout construction is presented.

The first step is to make a layout view using the SDL tool from Custom Compiler. All the used cells require properly defined pins and PR boundaries for each cell layout (as

```
module ctrl_init (EN, CLK, RST, INIT);

input EN, CLK, RST;
  output INIT;
  wire   N22, n3, n4, n5, n6, n7, n9, n10, n11, n12, n13, n14, n15, n16, n17,
         n18, n19, n20, n21, n22, n23, n24, n25, n26, n27 , vdd, gnd;
  wire   [1:0] state;

  DFFRNQN_0X_mod state_reg_0_ ( .D(n7), .CLK(CLK), .RN(n6), .Q(state[0]), .VDD(vdd), .VSS(gnd));
  DFFRNQN_0X_mod state_reg_1_ ( .D(n5), .CLK(CLK), .RN(n6), .Q(state[1]), .VDD(vdd), .VSS(gnd) );
  LATCH_0X INIT_reg ( .CLK(N22), .D(n3), .Q(n4), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U15 ( .A1(n26), .A2(n9), .Y(n7), .VDD(vdd), .VSS(gnd) );
  INV_4X U16 ( .A(n27), .Y(n9), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U17 ( .A1(n26), .A2(n10), .Y(n5), .VDD(vdd), .VSS(gnd) );
  INV_4X U18 ( .A(n25), .Y(n10), .VDD(vdd), .VSS(gnd) );
  NOR2_1X U19 ( .A1(n11), .A2(n12), .Y(n27), .VDD(vdd), .VSS(gnd) );
  INV_4X U20 ( .A(EN), .Y(n11), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U21 ( .A1(n13), .A2(n14), .Y(n12), .VDD(vdd), .VSS(gnd) );
  INV_4X U22 ( .A(state[0]), .Y(n14), .VDD(vdd), .VSS(gnd) );
  INV_4X U23 ( .A(state[1]), .Y(n13), .VDD(vdd), .VSS(gnd) );
  INV_4X U24 ( .A(EN), .Y(n15), .VDD(vdd), .VSS(gnd) );
  INV_4X U25 ( .A(state[0]), .Y(n16), .VDD(vdd), .VSS(gnd) );
  NOR2_1X U26 ( .A1(state[0]), .A2(n15), .Y(n17), .VDD(vdd), .VSS(gnd) );
  NOR2_1X U27 ( .A1(n15), .A2(n16), .Y(n18), .VDD(vdd), .VSS(gnd) );
  MUX2_0X U28 ( .IN0(n18), .IN1(n17), .S(state[1]), .Q(n25) );
  NOR2_1X U29 ( .A1(state[1]), .A2(n19), .Y(n24), .VDD(vdd), .VSS(gnd) );
  INV_4X U30 ( .A(EN), .Y(n19), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U31 ( .A1(n20), .A2(n21), .Y(N22), .VDD(vdd), .VSS(gnd) );
  INV_4X U32 ( .A(state[1]), .Y(n22), .VDD(vdd), .VSS(gnd) );
  INV_4X U33 ( .A(state[0]), .Y(n23), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U34 ( .A1(EN), .A2(n23), .Y(n21), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U35 ( .A1(n23), .A2(n22), .Y(n20), .VDD(vdd), .VSS(gnd) );
  INV_4X U36 ( .A(RST), .Y(n6), .VDD(vdd), .VSS(gnd) );
  INV_4X U37 ( .A(n4), .Y(INIT), .VDD(vdd), .VSS(gnd) );
  INV_4X U38 ( .A(n24), .Y(n3), .VDD(vdd), .VSS(gnd) );
  NAND2_2X U39 ( .A1(state[1]), .A2(state[0]), .Y(n26), .VDD(vdd), .VSS(gnd) );
endmodule
```

Figure 4.16: Verilog netlist for initiation logic circuit.

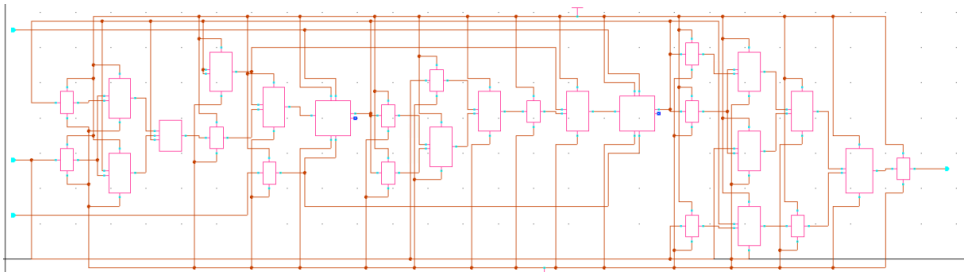


Figure 4.17: Schematic for initiation logic circuit.

presented in section 4.5.1). The placement tool can be selected once the SDL is loaded and the layout view is generated. The placement tool includes an auto-placement function which creates multiple options for placement for the given cells. This placement only considers area distributions and does not optimize any other factor, such as timing or interconnection distributions.

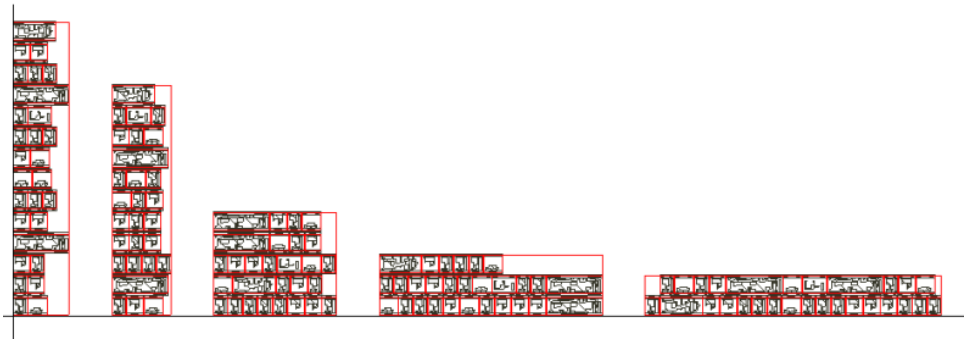


Figure 4.18: Placement gallery made with the placement tool in Custom Compiler for the initiation logic circuit.

The preliminary design should be copied to the layout view of the current circuit. Manual changes can be applied to the layout. After the placement changes have been finished, the Routing tool from Custom Compiler can be used in conjunction with the SDL tool. The SDL identifies each pin of the cells and the logic connections between each cell, similar to the netlist view. The routing tool uses the connections defined by the SDL to create metal interconnections between the pins while respecting DRC rules. Before applying the automatic routing option, it is recommended to define a routing area. The routing area limits the places the tool can use as routing options. Another variable that should be set is the metals used for routing. Without this setting, the tool can use any material available in the technology, including polysilicon which is not always the desired routing material. Figure 4.19 shows the final layout for the initiation logic circuit after using the automatic place and route techniques previously described.

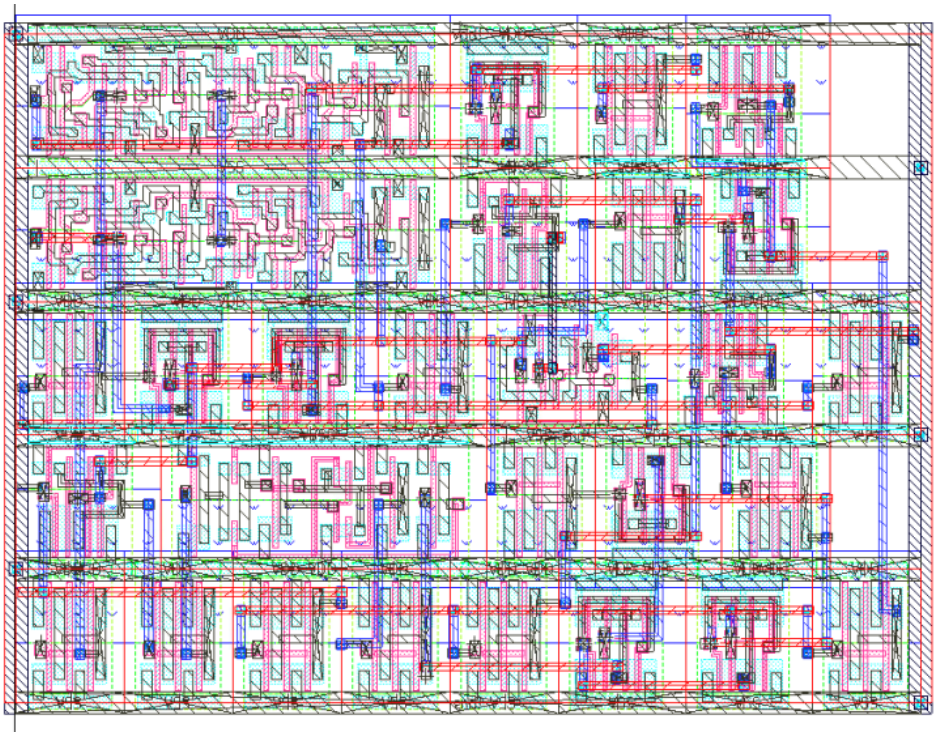


Figure 4.19: Layout view of the initiation logic circuit.

The custom place and route method have the downside that the final layout will not be the most efficient in terms of timing. Another negative aspect of this technique is that the routing tool does not take into account antenna rules. It becomes relevant in designs that have long interconnections between internal pins. To solve this specific issue, affected nets can be disabled in the automatic route. After that, the net can be manually connected to find a solution to reduce the length of the connection. It is not recommended for small designs (4 to 5 cells) to use the custom place and route method. In these cases, manual interconnections should be made in search of an area-efficient solution.

4.7 Complete base SRAM construction

This section presents the construction of the base SRAM block that was made by using the flows and methodologies presented on previous sections. Some of the design errors and its solutions are also be presented.

4.7.1 Custom SRAM routing

For the final SRAM layout the following blocks were used:

- SRAM64x32x2 [5]
- Decodificador6x64_EN_Latch [6]
- SRAM_control_circuit

The SRAM_control_circuit layout was based on the design proposed in [21]. The original version had a Verilog module with the instances of the custom cells needed. An additional FSM was added for an automatic initiation sequence for the control unit. The process of building the initiation sequence can be seen as an example for Section 4.6. Once all the layouts are ready and have the pins and PR boundaries properly defined, the auto-route tool of Custom Compiler is used to generate the interconnections between the three main blocks. Some manual changes had to be applied to the input data section since it ended being a dense interconnection area and the routing tool had issues finding a valid path. The top view layer of the SRAM after applying the interconnections can be seen in Fig. 4.20 and a zoom of the data input pass transistors in 4.21.

After passing the DRC and LVS checks, a parasitic extraction of the memory carried out for post-layout testing.

4.7.2 SRAM basic tests and read issues

A complete SRAM netlist with parasitic elements can represent a heavy computational weight when doing a SPICE simulation. A simple test can let the designers observe the general behavior of the SRAM. If an error appears in this early stage of testing, it means a design flaw is present in the memory. The basic test consists of three steps: initialization, write/read 0, and write/read 1. All of the sequences are made for a single word line to keep the simulation execution time at a minimum. For a thorough description of the signals involved in the control signal for each step refer to [21]. The waveform for the post-layout basic simulation can be found in Fig. 4.22. In this figure, an error is present during the reading process. The memory is reading a logic 0 no matter which logic value was written for the previous cycle.

After an extensive analysis of the reading process, it was determined that the error was because of the control logic applied for the sense amplifier and the wordlines enable signal. The control unit contains a couple of dummy cells which indicate an approximation of the

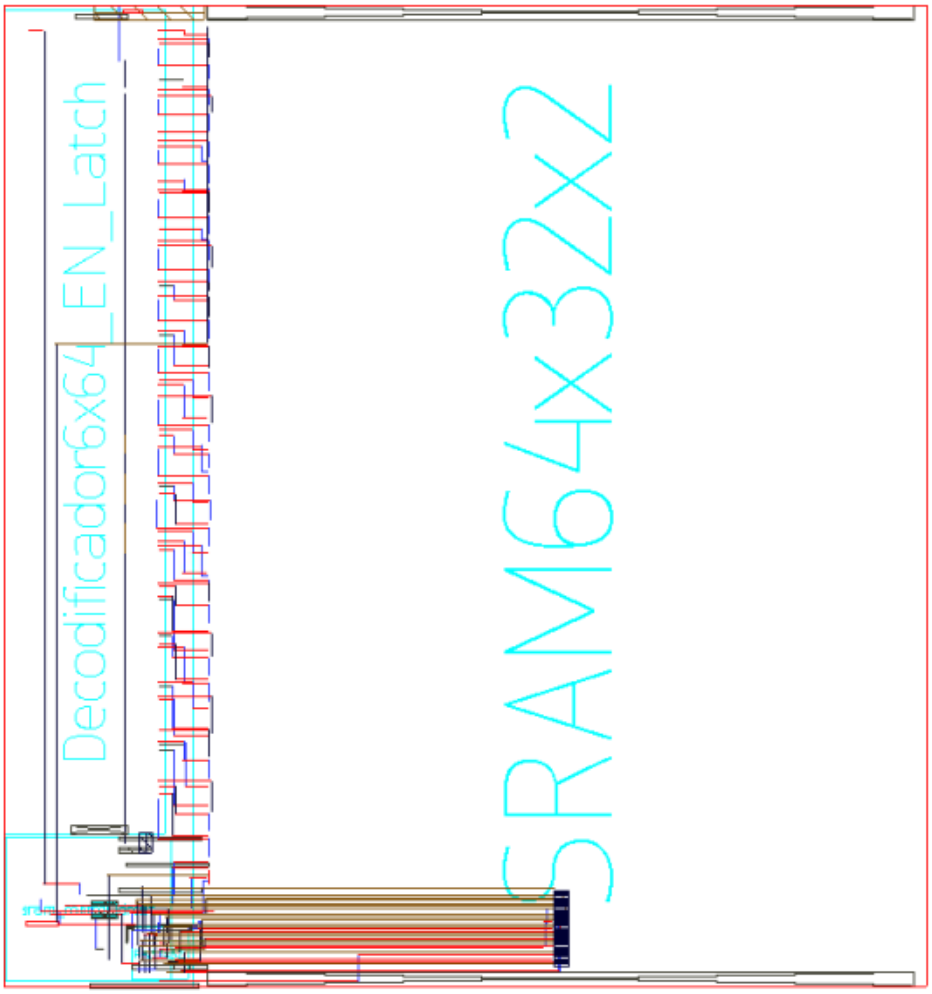


Figure 4.20: Preliminary complete SRAM layout.

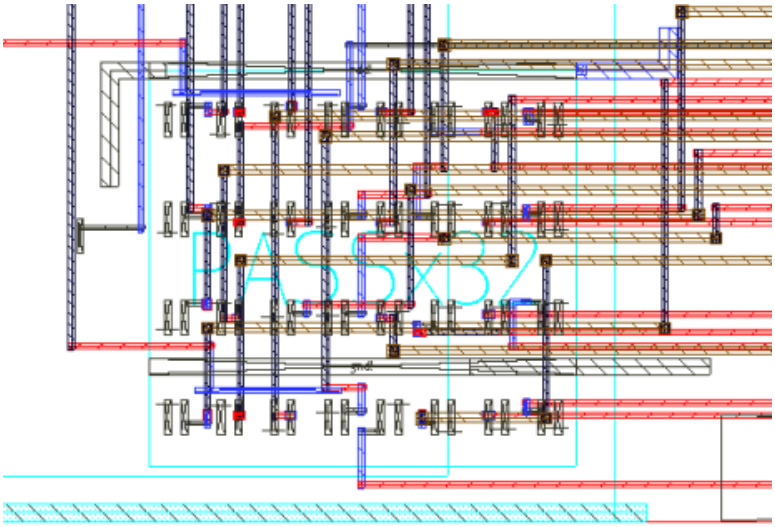


Figure 4.21: Interconnection of pass transistors for input data.

voltage values present on the bitlines. Once the dummy cells discharge the pseudo-bitlines,

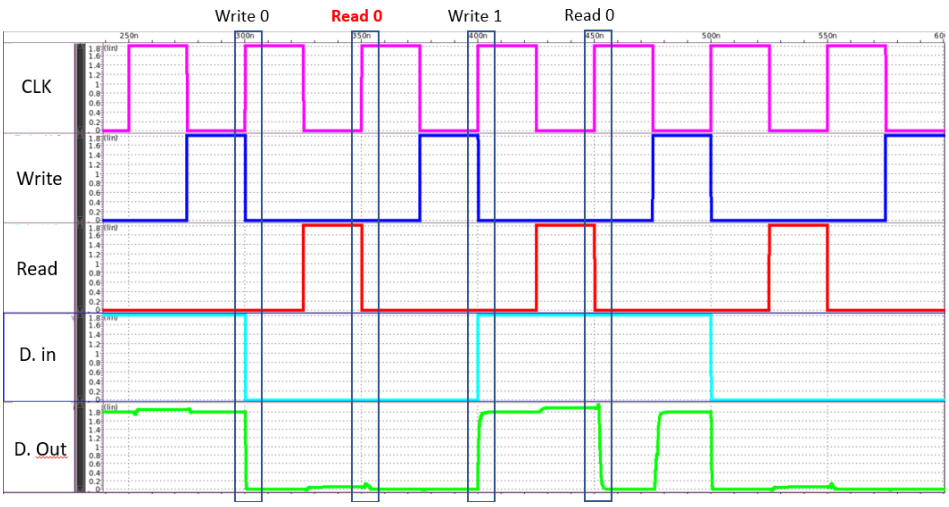


Figure 4.22: Post-layout simulation of preliminary complete SRAM with reading errors.

the control unit would close the wordlines by disabling the decoder enable signal. This would leave the bitlines with a voltage difference that the sense amplifier could detect. The main factor for this error was the time the enable signal would need to activate and deactivate the wordlines after all the interconnections and parasitic elements were present. The dummy bitlines discharged even faster than the flight time of the decoder enable signal, giving almost no time for the memory cells to be accessed.

This meant a change had to be made for either the decoder or the control unit. Due to the time constraints of the project, neither of those options was viable. Instead, a capacitor was added to the control unit to slow the internal signal that indicated that the wordline needed to be disabled. The detriment to this solution is that the benefits of the sense amplifier would be reduced since what the capacitor is doing is maintaining the wordline active for a longer period. At the schematic level, a sweep of capacitance values was made to obtain a window of 3 ns for the wordlines to be active. This ensures that the bitlines are discharged to a readable value in the post-layout simulations. The capacitance needed was 717 fF. Because of layout issues, the capacitance was achieved by constructing two parallel capacitors, one of 445 fF and the other of 272 fF. Fig. 4.23 shows the new layout for the control unit.

4.7.3 SRAM layout changes for physical library preparation

The preparation for the creation of a Milkyway library for the SRAM layout follows the same steps as a standard cell. Physical pins need to be defined, a PR boundary needs to be set, and an abstract view has to be extracted. With standard cells, the pin creation required that the pins matched certain spaces within the unit cells of the library. In the case of the SRAM, the analog circuits and complex interconnections limit the area in which a pin can be properly defined. To make the design route friendly, the input and output pins are created outside of the interconnection areas of the SRAM. This allows

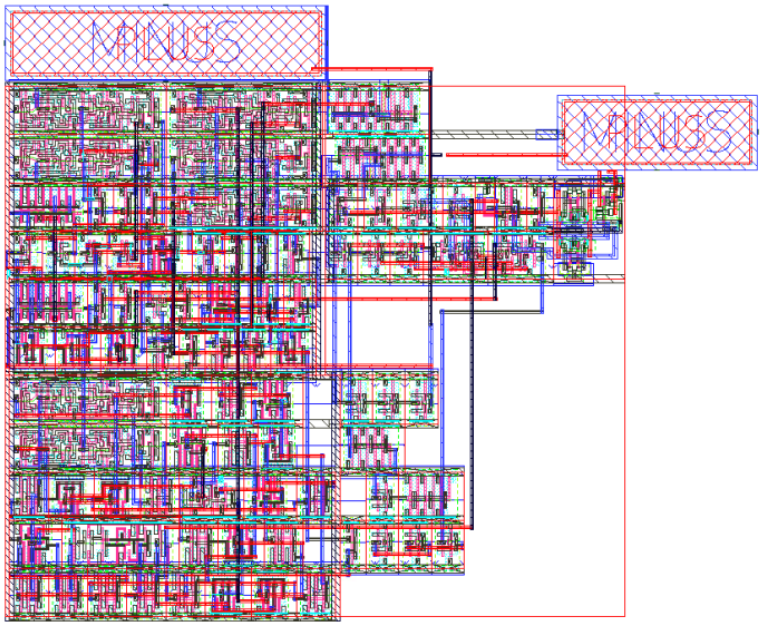


Figure 4.23: SRAM control unit with physical capacitors to fix reading issues.

the placement of pins to accommodate the design for different alignments or unit cells of other standard libraries.

Fig. 4.24 shows the final implementation for the input and output pins.

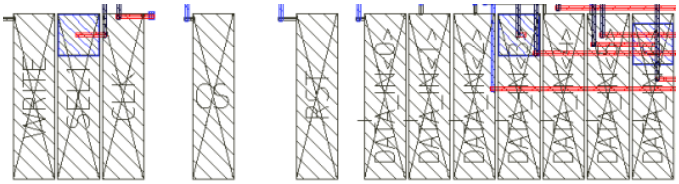


Figure 4.24: Example of physical pins for the SRAM layout

After defining the pins, the PR boundary can be set around the final area of the SRAM. Since this design is hierarchical, an additional setting needs to be defined when creating the abstract view. The abstract creation requires the number of hierarchical levels that are to be taken into account. The level selected needs to be at least the same level where the physical pins are created. If lower levels are taken into account, and more detailed abstract can be obtained, but the amount of time it requires to generate can be detrimental. Besides, the abstract view is used only for place and route, so detail of all the interconnections of a circuit is not required. Custom Compiler fills the areas belonging to lower hierarchical levels with blocks of the different metal layers included in the design, ensuring there are not any shorts when doing place and route of the memory. The final layout for the SRAM can be seen in Fig. 4.25 and Fig. 4.26 shows the abstract for the SRAM.

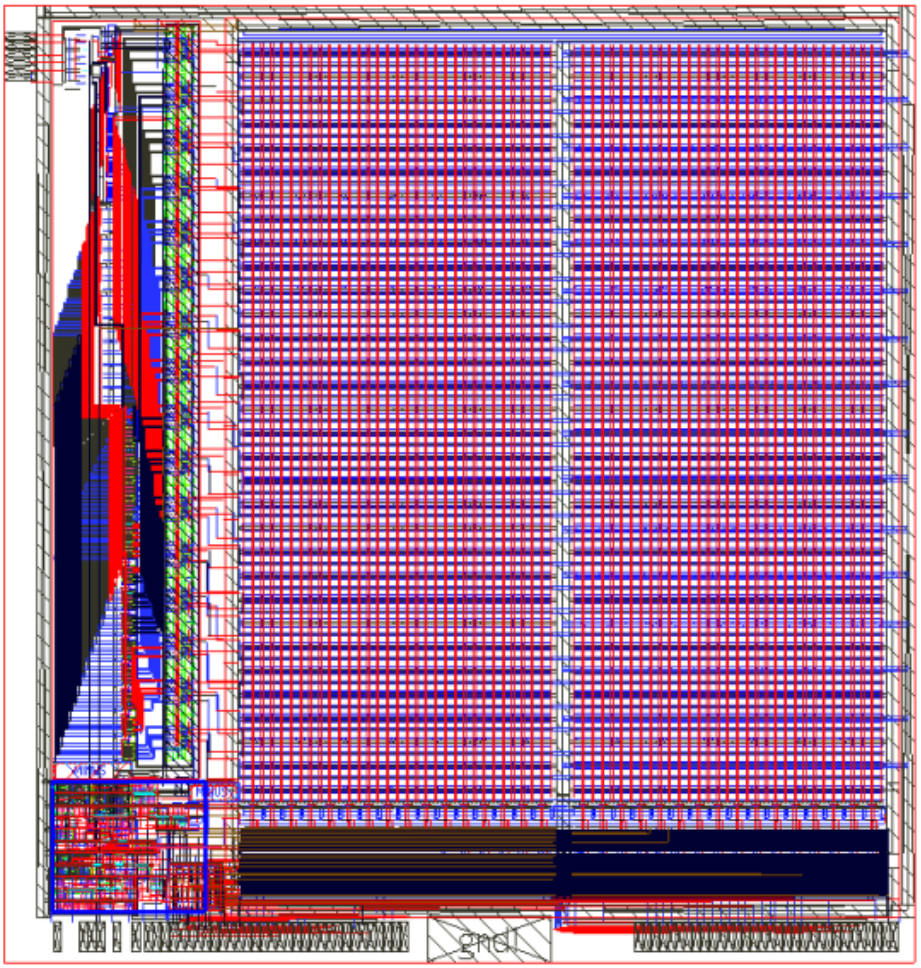


Figure 4.25: Final layout view for the complete 64x32x2 SRAM

The next steps to create the physical library are the same as the ones used for standard cells. The only change in the Milkyway script is in the contents of the Cell Type file. This SRAM design falls into the category of a black box.

4.8 Digital and physical synthesis for custom SRAM black box

This section contains the steps to follow to include a custom SRAM block into a digital synthesis flow. The test case presented synthesizes a complete 64x32x2 SRAM, 32 inverters for the output bus, and 32 flip-flops to capture the output bus values. This test creates timing paths that help verify the functionality of the SRAM library made in the prior sections and its compatibility with other standard cell libraries.

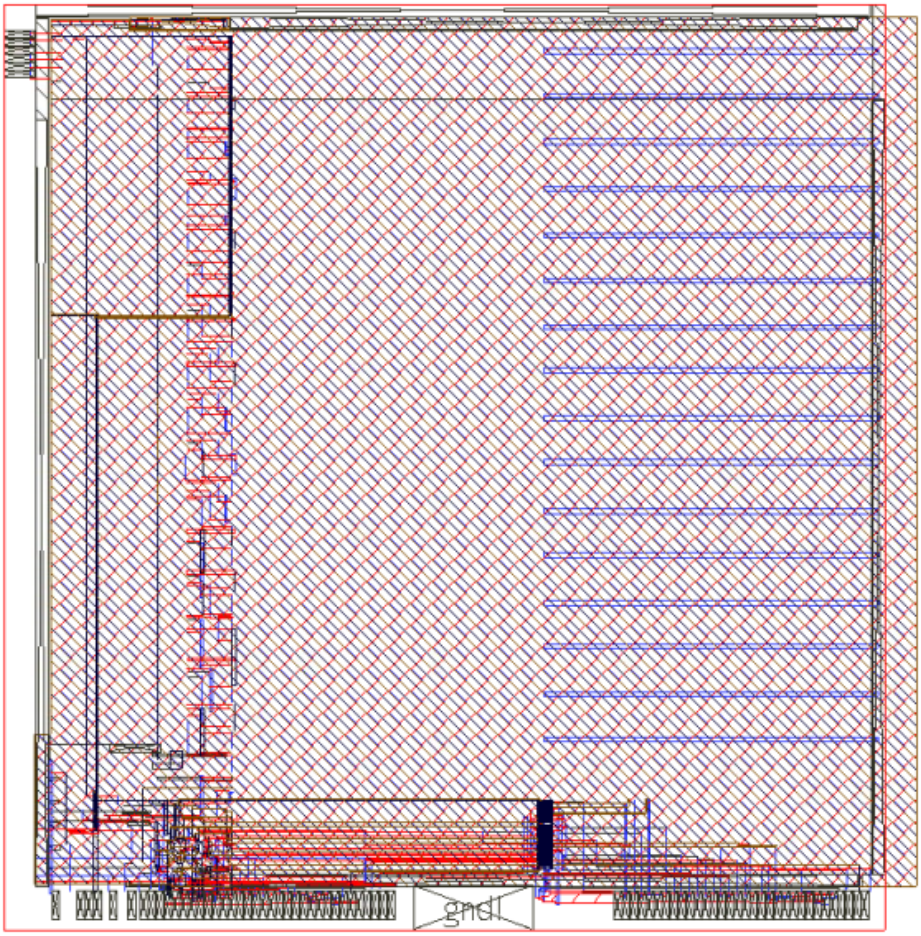


Figure 4.26: Abstract view for the complete 64x32x2 SRAM

4.8.1 Create Verilog file and digital synthesis

The first step in the synthesis process is to create a Verilog file. A module with the design name of the SRAM block includes the memory into the Verilog design. The SRAM module name has to match the name assigned to the memory characterized in the physical library. A regular Verilog black box only needs its ports defined and no logic or functionality is required. In Fig. 4.27 an example of the definition for the custom SRAM block is presented, including its instance in the main module.

The front-end flow used to set up the synthesis should not include the physical library for the custom SRAM. If the library is present, there will be an error with the black box creation since the library only has physical information and does not have any timing data yet. After the synthesis, the SRAM black box module should still be visible in the synthesized Verilog. A section of the schematic view made after the front-end flow is presented in Fig. 4.28

```
module c_sram_64x32x2_cambios_en_netlist (CS, R, RST, SEL1, W, CLK, DIR, DATA_IN, DATA_BUS);

  input CS;
  input R;
  input RST;
  input SEL1;
  input W;
  input CLK;
  input [0:5] DIR;
  input [31:0] DATA_IN;
  output reg [0:31] DATA_BUS;

endmodule


module SRAM (CS, R, RST, SEL1, W, CLK, DIR, DATA_IN, DATA_BUS_INV);

  input CS;
  input R;
  input RST;
  input SEL1;
  input W;
  input CLK;
  input [0:5] DIR;
  input [31:0] DATA_IN;
  output reg [0:31] DATA_BUS_INV;

  wire [0:31] inv;
  wire [0:31] DATA_BUS;

  c_sram_64x32x2_cambios_en_netlist memory_0(.CS(CS), .R(R), .RST(RST), .SEL1(SEL1), .W(W), .CLK(CLK), .DIR(DIR), .DATA_IN(DATA_IN), .DATA_BUS(DATA_BUS));

  assign inv = ~DATA_BUS;
  always @(posedge CLK)
  begin
    DATA_BUS_INV <= inv;
  end

endmodule
```

Figure 4.27: Verilog example for SRAM black box module and instance

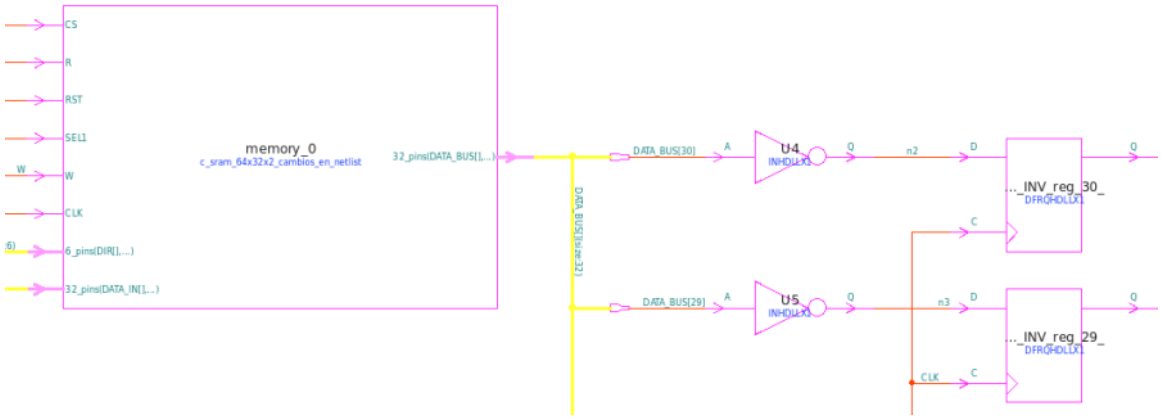


Figure 4.28: Zoom in of schematic view for Verilog netlist showing SRAM block connected to additional output logic from other libraries.

4.8.2 Quick time model (QTM) creation for SRAM black box

The SRAM design has a higher logic and timing complexity than the standard cells characterized through Silicon Smart. Silicon Smart can characterize memory circuits but, the tool requires a specific set of input and output ports to work. This limits the types and variations of SRAM that can be characterized in this way. The QTM approach provided by ICC serves as a workaround. QTM is a series of commands that create a valid logic library for ICC. The timing values and constraints are added manually, requiring an exhaustive knowledge and testing of the circuit to obtain valid and realistic timing models. The following is a list of items that need to be included in the QTM for the SRAM modules:

- Design physical dimensions.

- Global setup and hold constraints (base value applied to all pins, in nanoseconds).
- Input and output pin name definitions.
- Clock pin definitions.
- Load estimate for input and clock pins (fF).
- Arc delay between clock pin to output pin (block delay in nanoseconds).
- Additional setup/hold values for specific input pins.

The output of the QTM flow is a logic library that only ICC can read. ICC can be restarted after the library is created. The new library is then added to the link library list. If the QTM flow is carried out correctly, the synthesized Verilog design can be loaded and the linking of the libraries should be successful.

4.8.3 Physical synthesis of design with custom SRAM block

The first step to the physical design is the creation of the floorplan. After successfully adding the logic library of the custom SRAM, ICC reads the SRAM module the same way as it would read a standard cell. A floorplan blockage needs to be created around the SRAM module. This blockage keeps other instances from being placed over or near the cell and its pins. An example of the floorplan generated is seen in Fig. 4.29.

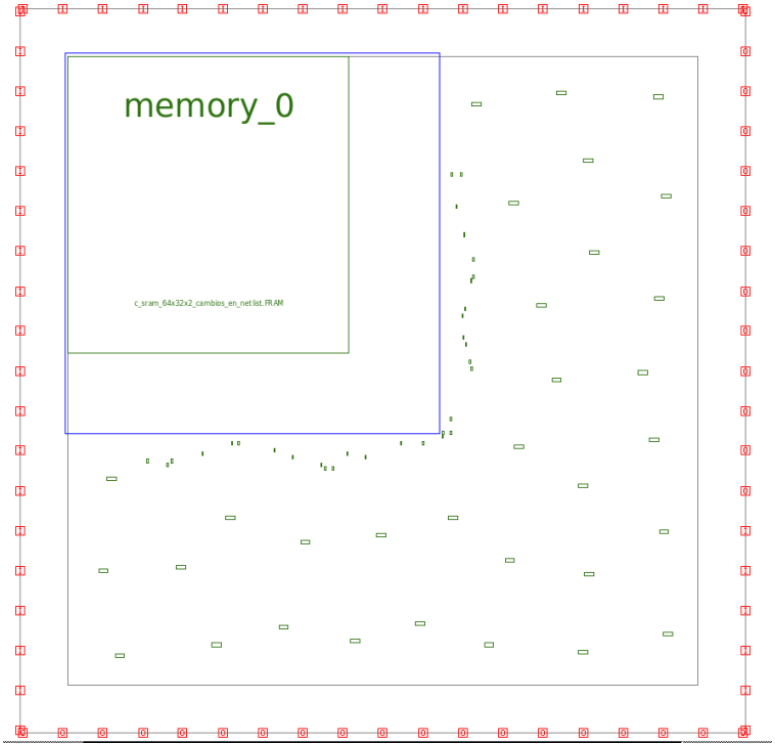


Figure 4.29: Floorplan including SRAM instance, FP blockage and additional output logic instances

When the cells have been placed, the power grid can be generated. Here, one can check if the physical library of the memory has been generated correctly. Checking the abstract view of the design in Fig. 4.26, all of the metal layers, except the highest layer,

are blocked. In Fig. 4.30 the power grid has been synthesized, and as expected, only the highest metal layer is crossing the SRAM block.

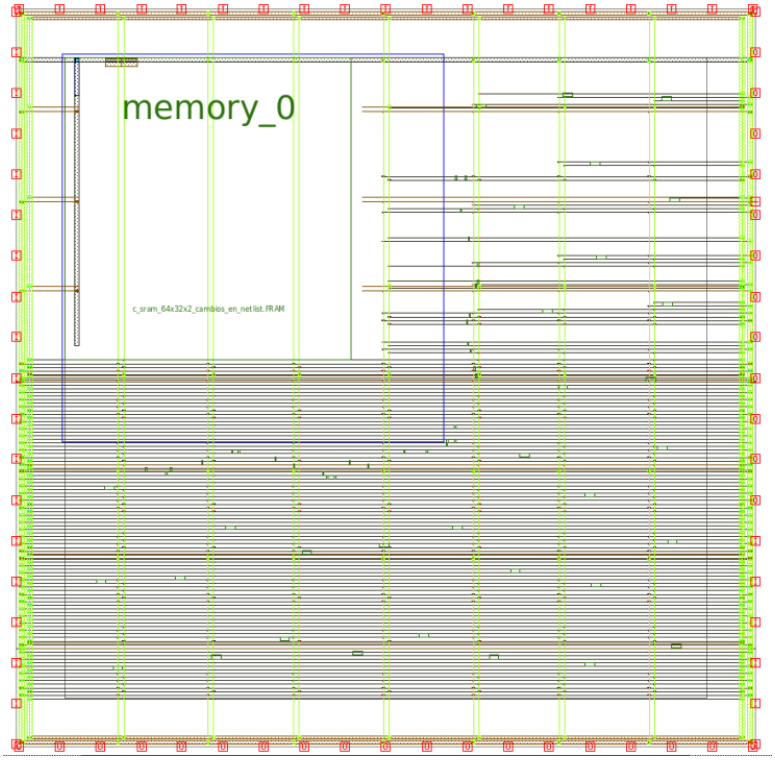


Figure 4.30: Powerplan synthesized. The grid respects the blockage of the SRAM cell.

The next step in the physical synthesis is the placement of fill cells. Again, these cells respect the FP blockage created for the SRAM, leaving enough space for the correct connection of all the pins and not placing any other cell inside the SRAM cell. Once the fill placement finishes, the clock tree is synthesized. Here, the designer can verify if the tool is reading the pin information for the custom block. In Fig. 4.31 the clock tree has been synthesized and can be seen connecting the different output flip-flops and the SRAM.

The final major step is to route data signals and nets of the design. The output and input pins of the SRAM should connect to the ports of the design or pins of other cells without issue. The final layout of the routed design can be seen in Fig. 4.32. To verify that not only the physical connections are being made, but to also check the logic library created, the timing report can be generated. Since the specific test case shown uses flip-flops to capture the output of the SRAM, the timing path has the constraints defined during front-end synthesis, delays added by the standard cells, and the delay set in the QTM model. The timing path should also reflect the correct behavior of the SRAM. The correct timing behavior can be seen in Fig. 4.33.

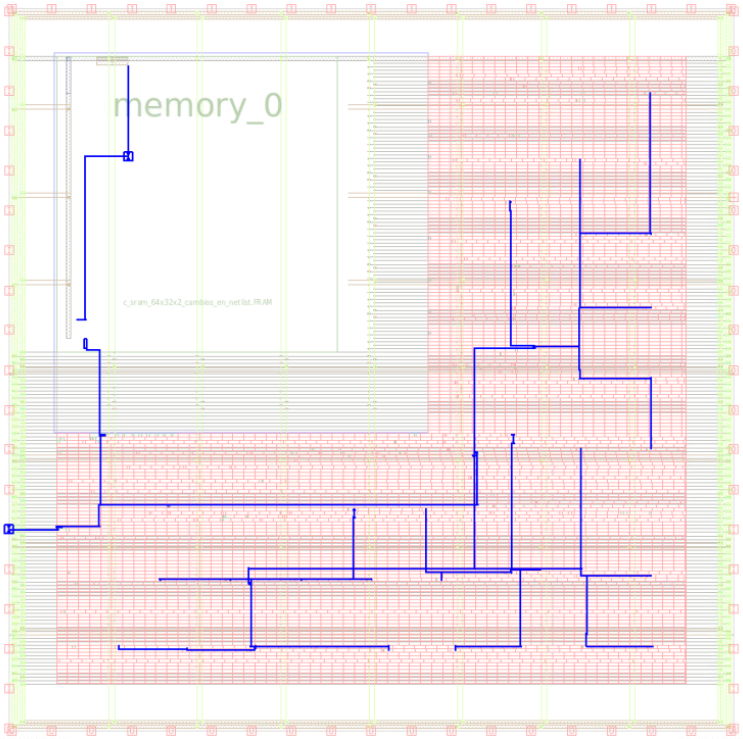


Figure 4.31: Synthesis of the clock tree. Clock signal is successfully connecting to the SRAM cell.

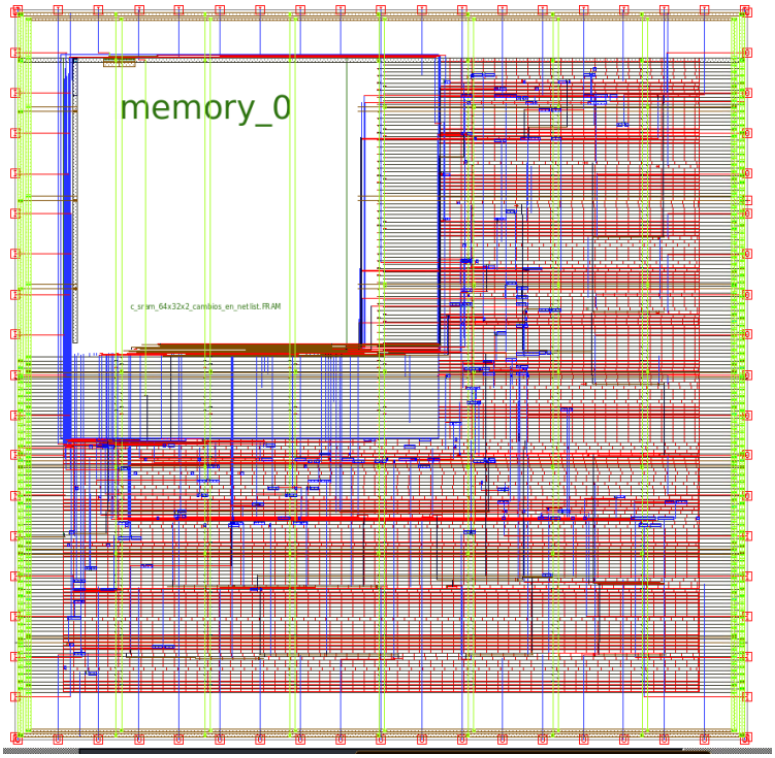


Figure 4.32: Final layout view for physical synthesis of a design with a custom SRAM block.

```
Startpoint: memory_0/CLK
            (internal path startpoint clocked by CLK)
Endpoint: DATA_BUS_INV_reg_20_
          (rising edge-triggered flip-flop clocked by CLK)
Path Group: CLK
Path Type: max

Point                                                    Incr       Path
-----------------------------------------------------------------------------
clock CLK (fall edge)                                   25.00      25.00
clock network delay (propagated)                         3.12      28.12
input external delay                                     0.00      28.12 f
memory_0/CLK (c_sram_64x32x2_cambios_en_netlist)         0.00      28.12 f
memory_0/DATA_BUS[20] (c_sram_64x32x2_cambios_en_netlist)
                                                         5.50 @    33.62 f
U14/Q (INHDLLX0)                                         1.22 @    34.85 r
DATA_BUS_INV_reg_20_/D (DFRQHDLLX1)                      0.01 &    34.85 r
data arrival time                                                  34.85

clock CLK (rise edge)                                   50.00      50.00
clock network delay (propagated)                         3.21      53.21
clock uncertainty                                       -0.50      52.71
DATA_BUS_INV_reg_20_/C (DFRQHDLLX1)                      0.00      52.71 r
library setup time                                      -0.20      52.51
data required time                                                 52.51
-----------------------------------------------------------------------------
data required time                                                 52.51
data arrival time                                                 -34.85
-----------------------------------------------------------------------------
slack (MET)                                                        17.65
```

Figure 4.33: Timing reports for a single SRAM data bus output path. Slack is met for a 50 ns period clock.The expected SRAM behaviour is also confirmed, the output data is launched and captured for a half cycle.

Chapter 5

Simulation results and analysis

This chapter contains tests applied to the complete SRAM 64×32×2 array. This includes the decoder and control unit. All of the designs are presented at a post-layout level. A section is also dedicated to the impact of the automatic router in Custom Compiler and the changes made to the control unit in order to solve the reading bug.

5.1 Control Unit Post-Layout analysis

The control unit presented in this work was one of the modules that required more work since a complete layout for the design was not available. The complexity of this design did not allow for a practical way to create a layout manually. Besides, as seen in previous chapters, an error in the timing of the output signal of this block can generate bugs during read and write accesses. In the Table 5.1 a comparison is made between the base control unit presented in [21] and the final layout of this document. To make the test as fair as possible, the complete layout for both the control unit and the SRAM block are being used, but the connections between both are ideal.

For write cycles the main control signals are ENW (enable input pass transistor for write) and END (enable decoder). In the original design both signals activate close to

Table 5.1: Control unit main output signal timing.

Measurement	Base [21]	Presented work
$t_{CLKr} \rightarrow$ ENW (ns)	0.833	0.923
$t_{CLKr} \rightarrow$ EN_BIT (ns)	1.42	1.62
$t_{CLKr} \rightarrow$ ENDW (ns)	0.821	1.8
$t_{CLKr} \rightarrow$ ENDR (ns)	1.02	1.92
$t_{CLKr} \rightarrow$ SAE (ns)	1.96	2.01
$t_{CLKr} \rightarrow$ NSAE (ns)	1.97	2.59
t_{PENDR} (ns)	0.695	3.52

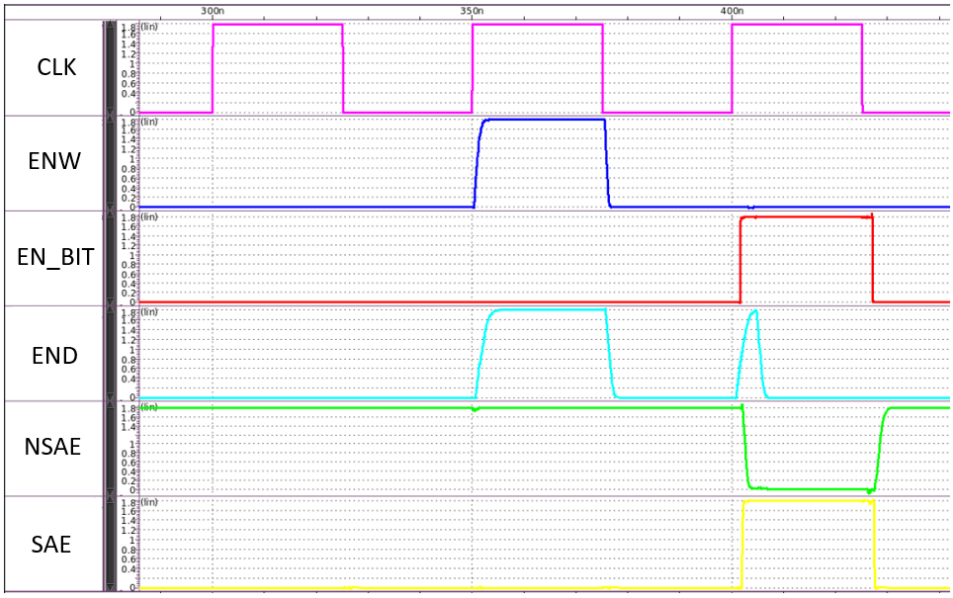


Figure 5.1: Presented work control unit waveform for main signals

each other. Activating the decoder before the input is driving the wordline can cause unnecessary power consumption since the bit cell can move to a different state than the one that is going to be written. In the final layout the decoder signal activates almost 1 ns after the input is being driven into the bitlines. This behaviour reduces this unwanted power consumption but also makes the writing process more reliable.

In the case of read cycles the relevant signals are EN_BIT (first access between bit lines and sense amplifier), SAE (sense amplifier enable) and END. The first expected signal is END, allowing the bit lines to start transitioning to the data inside the bit cell. After that EN_BIT sets the sense amplifier into its active state. Finally, SAE and NSE dictate the start of the driving process of the sense amplifier read data into the output bus. In the base design, END accesses the bit cells before the EN_BIT starts the reading process. In the presented work EN_BIT activates even before the cells have been accessed, creating a reading error. That is why the active time for the END signal during read was extended, giving more time to the bit cells to drive the whole reading system and fix the initial wrong value set in the sense amplifier. This effectively turns the read into an regular read process and not a small signal reading process. Although power consumption increases due to this fix, the reliability of the reading process also increases by having a higher voltage difference in the sense amplifier input nodes. All of this was caused because the output capacitance that END has to drive was given an overly optimistic value.

The difference seen between SAE and NSAE adds additional delay during the read of a logic 1 but does not affect the overall reliability of the reading process.

5.2 Validation of read behaviour through analog simulations

A series of analog simulations where set to ensure that the changes applied to the control unit solve the reading issues. To obtain a more detailed behaviour of the memory multiple corners where tested. The corners selected are the following:

- Typical
- Worst one (fast n / slow p)
- Worst zero (slow n / fast p)
- Worst power (fast n / fast p)
- Worst speed (slow n / slow p)

All of the used corners operate at 25 °C and 1.8 V. These simulations are made using the complete SRAM circuit with post-layout parasitic information included. In Fig. 5.2 the waveform for the typical corner is presented.

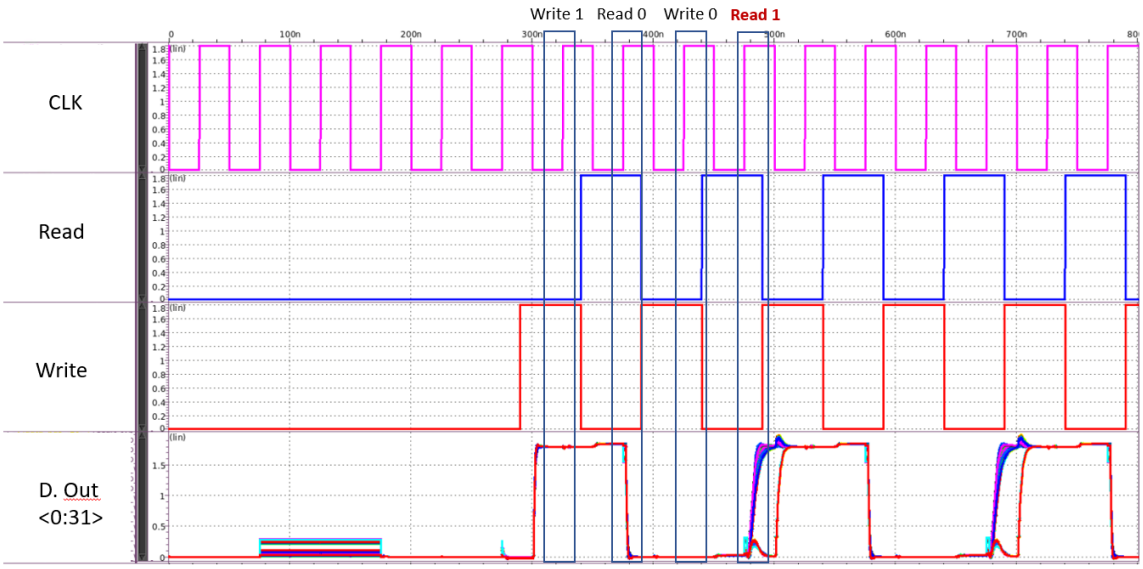


Figure 5.2: Complete post-layout SRAM simulation using typical corner. WL 0 was accessed and during Read 1 operation bits 4, 9, 14, 18, 23, 27, and 31 failed

For corners WO, WP, and WS the error is present on different sets of bits. Although each corner runs with the exact same inputs, it is possible that the initial values for other cells in the memory array to have different initial values. This differences can create different leakage scenarios for each bit line. This can explain the reason why each corner is producing different errors but it does not provide an answer to why the read one process fails for every corner.

In order to search for this answer Table 5.1 will be used as first guide. It was previously mentioned that the EN_BIT signals needs to be activated after the END signal during reading accesses. In Fig. 5.3 the schematic of the used sense amplifier is shown. This sense amplifier has an inner feedback loop between two inverters. This feedback loop is

forced to an illegal state by closing the connection to ground and forcing a logic one to both sides while EN_BIT is off. Once EN_BIT turns on, the feedback loop proceeds to enter an stable state. The values on the bitlines dictate which side enters a logic 0 state first, maintaining the logic one on the other inverter. Since EN_BIT activates before the decoder is enabled, the sense amplifier will begin to set its value using the precharge state of the bitlines. In the cases where the reading issue is present, the feedback loop enters an stable state in which the inner cells don't have enough time to flip the loop into the correct value.

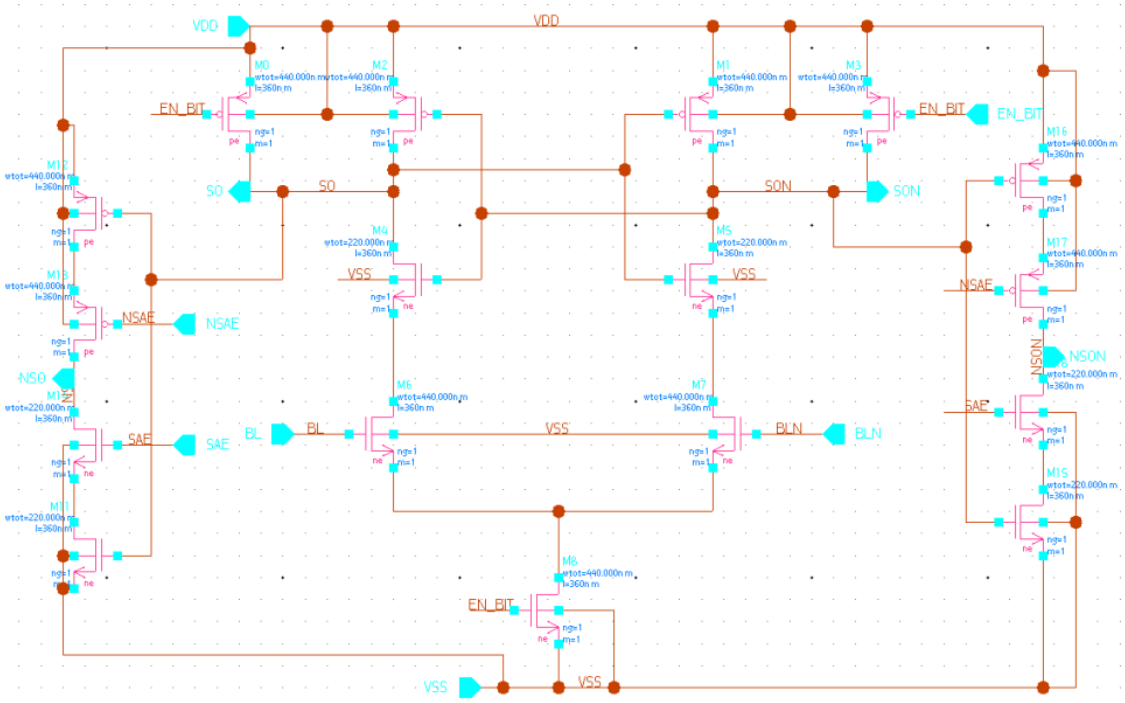


Figure 5.3: Sense amplifier schematic

Corner WZ presents a different error from the rest of the corners. The waveform for WZ corner is shown in Fig. 5.4. In this specific case, read access in not done by any of the bits. An slight voltage change can be seen during the read access, but it is not enough to represent the expected logic value. Upon further investigation of this issue, the control unit was found to be failing. Fig. 5.5 shows an equivalent analysis than the one shown in 5.1 but using WZ corner.

For reading cycles, the control unit uses two dummy SRAM cells to emulate the read timing of the cells in the memory array. This behaviour should help properly time the control signal to successfully enable and disable the sense amplifier. Fig. 5.6 shows the inner section of the control unit that contains the dummy cells.

First the dummy cell is set to a defined state with a logic 0 on the bit port. Once a read access starts, the dummy cell is also accessed, activating three different signals: EN_BIT, SAE and NSAE. As seen in Fig. 5.5, none of these three signals is activating during a read cycle. To ensure that the inner cell of the dummy cell is the issue an additional simulation was created under the current WZ corner. Fig 5.7 has the results for this test.

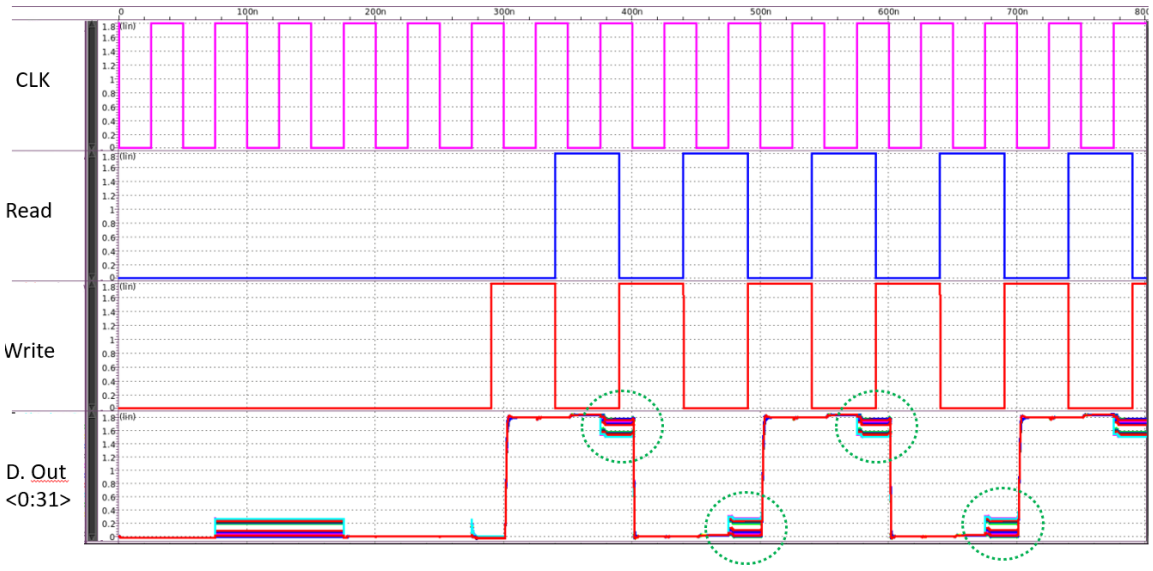


Figure 5.4: Read access error for WZ corner. Circled section shows incomplete transitions for both read logic 1 and read logic 0

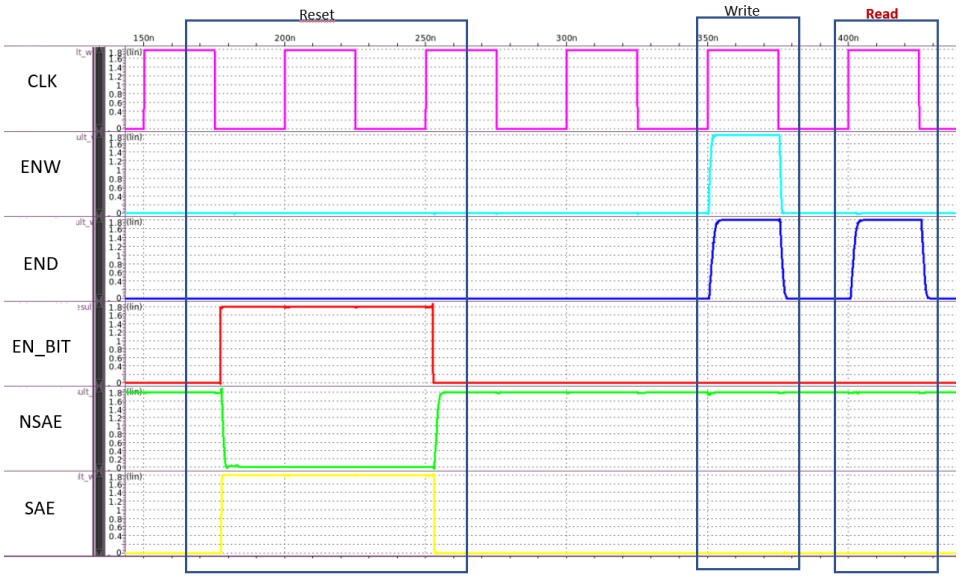


Figure 5.5: Control unit error for WZ corner. Read signals work during reset phase but do not activate during read cycles. Decoder enable does work for both read and write cycles.

To finish the analysis of the corner simulations, Table 5.2 presents the write access times and the read access times for the working bits.

All of the access 1 cases show to be the critical case for all the available corner results. As mentioned in section 5.1, the difference between SAE and NSAE signals creates an additional delay when setting a logic 1 into the output bus. This values can be lowered if the delay time of both signals is reduced. Even for a big signal read circuit, the read data should be ready in less than 5 ns for this design (see results of [4] and [5]). Most of

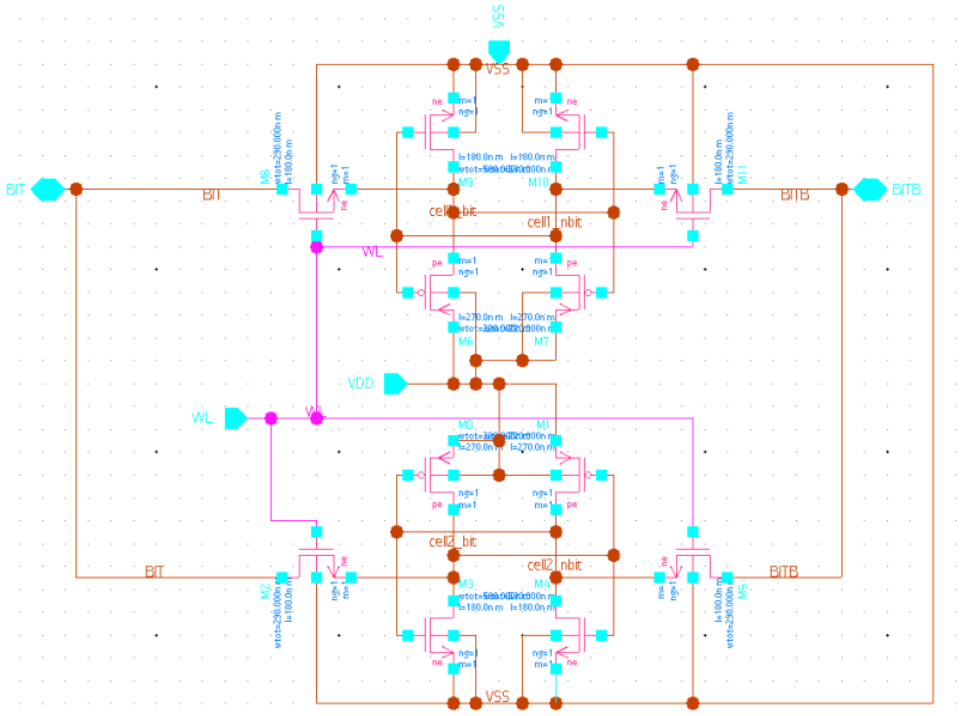


Figure 5.6: Schematic for dummy SRAM cell. This specific design consist on a double SRAM cell.

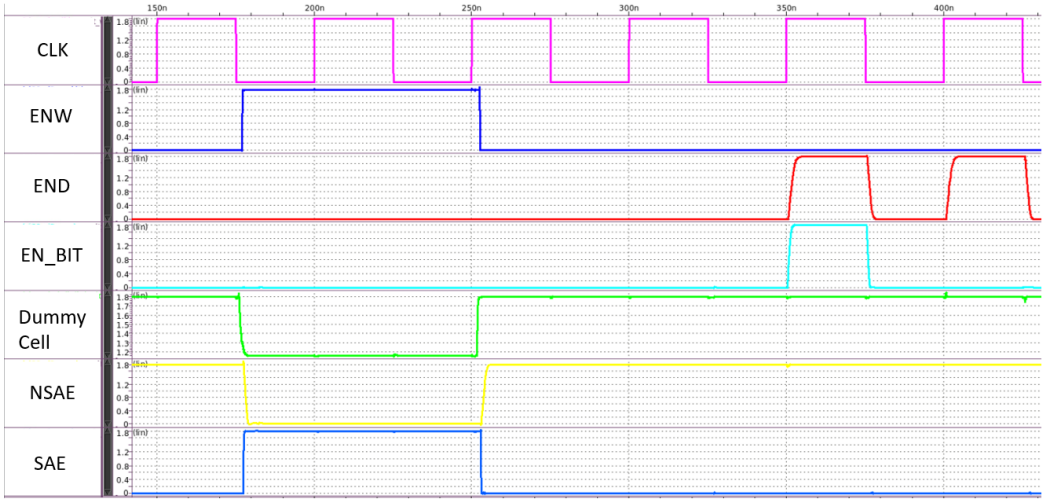


Figure 5.7: Control unit error for WZ corner with dummy signal included. As suspected, the dummy cell is not retaining the logic 0 value after the reset of the memory.

the delay is being contributed by the control unit signal timing and the wrong expected load values set during the control unit design stages.

Table 5.2: Different corners read access time

Corner	Access time 0 (ns)	Access time 1 (ns)
Typ	2.79	6.6
WO	2.9	21.3
WZ	N/A	N/A
WP	2.07	4.82
WS	3.74	17.2

5.3 FineSim mixed signal simulation

SRAM designs require analog simulations to verify the functionality of the design and to obtain a proper characterization. Analog simulations limit the type of test patterns that this type of memory requires. Another negative aspect of the analog simulation is that the time per test is high in complex netlists, such as the ones that have parasitic elements. Mixed-signal simulations are a solution to these issues, allowing to have the versatility of and speed of digital stimulus and precision of results on specific elements of the design.

FineSim is a Synopsys tool that provides time-efficient simulations for complex layouts. The simulations can be set to have different levels of accuracy depending on the desired tests. For the tests applied in this work, the option "spicexd" was used. The tools manual indicates that this mode is used specifically for a post-layout design that requires a digital stimulus.

The extracted layout requires some changes to be used in FineSim. In Fig. 5.8 the used modifications are presented. Additionally, HSPICE commands can be used to obtain analog measurements, as seen in Fig. 5.9.

```
**FineSim options
.option finesim_mode=spicexd
.vec stimulus.vec
.option finesim_vector_mode=1

**Analog voltage sourve
v7 vdd! gnd! dc=1.8

**Technology options
.temp 25
.lib '/mnt/vol_NFS_Zener/WD_ESPEC/falberto/ambiente_custom_mixta/tutorial_design/cells_1/Hspice/lpmos/xh018.lib' tm
.lib '/mnt/vol_NFS_Zener/WD_ESPEC/falberto/ambiente_custom_mixta/tutorial_design/cells_1/Hspice/lpmos/param.lib' 3s
.lib '/mnt/vol_NFS_Zener/WD_ESPEC/falberto/ambiente_custom_mixta/tutorial_design/cells_1/Hspice/lpmos/config.lib' default
```

Figure 5.8: FineSim analog options

```
.tran 10p 1550n start=0

*.probe tran v(*) level=1
.probe tran v(clk) v(cs) v(rst) v(sel1) v(r) v(w) v(dir<0>) v(data_in<0>) v(data_bus<0>) i(v7)
i0|i0|i36|i6|i1|i5|i1|m4|x1|src)

.option PARHIER = LOCAL
.option WARN = 0
.option finesim_soa_warn = 0
```

Figure 5.9: HSPICE option examples

To set the digital stimulus FineSim gives the option to manually write a table with

the values that the input signals will present during set intervals of time. A detailed explanation for the specific sequence of inputs required for read and write cycles can be found in [21]. The test applied consists of write cycles followed by read cycles over the same word to obtain access times and dynamic power consumption per access. As an additional option to check for errors, another column can be added to the table for expected output values. This option is used during the execution of read for the output data bus. A sample of the test table can be seen in Fig. 5.10.

```
;  Vector Pattern Definition
radix     1   1  1   1   1 1  111111   44444444        44444444
nodename clk CS RST SEL1 R W DIR<0:5> DATA_IN<31:0> DATA_BUS<31:0>
io        i   i  i   i   i i  iiiiii   iiiiiiii        oooooooo

; Mask function for specific signals
mask mask1 clk
mask mask2 DATA_BUS<31:0>

; Check window for output signals
check_window -0.5 0.5 2 25 393 mask2

; Waveform parameter setting
Tunit 1n
Slope 100p
VIH 1.8
Trise 1n mask1
Tfall 1n mask1

; Tabular Data
period 25;
; clk cs rst sel1 r w   dir    data_in   data_bus(valor esperado)
  0   0   1   0   0 0 000000 00000000    XXXXXXXX
  1   0   0   0   0 0 000000 00000000    XXXXXXXX
  0   0   1   0   0 0 000000 00000000    XXXXXXXX
  1   0   1   0   0 0 000000 ffffffff    XXXXXXXX
  0   0   1   0   0 0 000000 ffffffff    XXXXXXXX
  1   0   1   0   0 0 000000 ffffffff    XXXXXXXX
  0   0   1   0   0 0 000000 ffffffff    XXXXXXXX
  1   0   1   0   0 0 000000 00000000    XXXXXXXX
  0   1   1   1   0 0 000000 00000000    XXXXXXXX
  1   1   1   1   0 0 000000 00000000    XXXXXXXX
  0   1   1   1   0 0 000000 00000000    XXXXXXXX
  1   1   1   1   0 0 000000 ffffffff    XXXXXXXX
  0   1   1   1   0 0 000000 ffffffff    XXXXXXXX
  1   1   1   1   0 1 000000 ffffffff    XXXXXXXX
  0   1   1   1   0 1 000000 ffffffff    XXXXXXXX
  1   1   1   1   1 0 000000 00000000    XXXXXXXX
  0   1   1   1   1 0 000000 00000000    00000000
```

Figure 5.10: Section for test table file for digital signals in FineSim test

As an example of FineSim correctly detecting errors, a wrong value was introduced to the verification table during one of the read cycles. In Fig. 5.11 the results of the detected error are shown. If no errors are detected for a given table, the file will not be generated.

5.4 SRAM simulation results

As mentioned before, the first test intends to give an insight into the general characteristics of the designed SRAM block. In Fig. 5.12 a section of the test is presented. The write cycle times are measured from the positive clock slope to the change in the inside of the memory cell. Read cycle times are measured from the positive clock slope to the

Time	Signal	Simulated	Expected
518.500ns	data_bus<31>	1	0
518.500ns	data_bus<30>	1	0
518.500ns	data_bus<29>	1	0
518.500ns	data_bus<28>	1	0
518.500ns	data_bus<27>	1	0
518.500ns	data_bus<26>	1	0
518.500ns	data_bus<25>	1	0
518.500ns	data_bus<24>	1	0
518.500ns	data_bus<23>	1	0
518.500ns	data_bus<22>	1	0
518.500ns	data_bus<21>	1	0
518.500ns	data_bus<20>	1	0
518.500ns	data_bus<19>	1	0
518.500ns	data_bus<18>	1	0
518.500ns	data_bus<17>	1	0
518.500ns	data_bus<16>	1	0
518.500ns	data_bus<15>	1	0
518.500ns	data_bus<14>	1	0
518.500ns	data_bus<13>	1	0
518.500ns	data_bus<12>	1	0
518.500ns	data_bus<11>	1	0
518.500ns	data_bus<10>	1	0
518.500ns	data_bus<9>	1	0
518.500ns	data_bus<8>	1	0
518.500ns	data_bus<7>	1	0
518.500ns	data_bus<6>	1	0
518.500ns	data_bus<5>	1	0
518.500ns	data_bus<4>	1	0
518.500ns	data_bus<3>	1	0
518.500ns	data_bus<2>	1	0
518.500ns	data_bus<1>	1	0
518.500ns	data_bus<0>	1	0

Figure 5.11: Example of error file generated by FineSim

Table 5.3: Complete SRAM of 64×32×2 results

Area (μm^2)	Read access (ns)	Write access (ns)	Dynamic power read (μW)	Dynamic power write (μW)	Dynamic power precharge (μW)	Dynamic power cycle (μW)
133,887.065	5.92	2.87	603	346	1185	895

change on the output data bus. In the case of dynamic power, three different results are presented: write average power, read average power, and precharge average power, taken from samples of 25 ns (half a clock cycle). Finally, the average power for a complete clock cycle is annotated for comparison with other works. This result is measured from a 50 ns time. Table 5.3 shows the worst results reported over different measurements in different cells and memory directions. Although there are more pessimistic scenarios, these results show a general overview of the expected behavior.

These results reveal important information and issues regarding the tested design. The first issue can be seen through the read dynamic power. In theory, the sense amplifier should be minimizing the ΔV occurring in bit lines during the read cycles. Because of inaccuracies during the design of the control circuit the wordline is not closing as fast as it is required. Another point of concern regarding the power consumption is the number

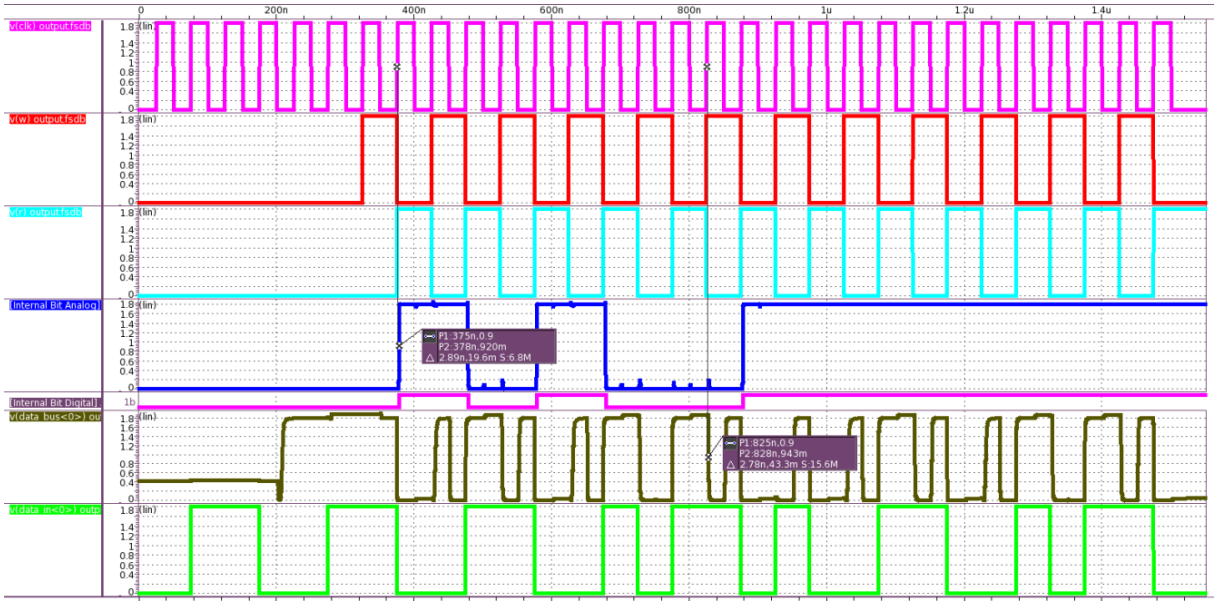


Figure 5.12: Section of resulting waveform of complete SRAM of 64×32×2 created through FineSim

of changes present in the output data bus. The physical design of these lines is almost as long as the width of the memory, becoming an element of dynamic consumption that cannot be ignored.

5.5 State of the art comparation

The last point of analysis for the results taken is the comparison with other SRAM arrays, presented in Table 5.4. For a fair contrast between results, simulations are indicated whether done at the schematic (S) or layout (L) levels. The scaling techniques presented on [22] are used for designs lower than 180 nm. All the results are scaled to a 180 nm technology with a V_{DD} of 1.8V.

Table 5.4: Comparison of SRAM array designs

Work	A. Size	Cell type	Tech. (nm)	V_{DD}	Access time (ns)	Dynamic power (μW)
[23]	4 Kb	10T S	90	0.3	69.41	292.92
[24]	2 Kb	6T L	65	1.2	NA*	87.95
[25]	10 Kb	6T L	130	1.5	0.64	NA*
[9]	16 Kb	10T L	90	0.4	160.37	66.03
[12]	1024 b	6T S	180	1.8	13.29	1090
Presented Work	**4096 b**	**6T L**	**180**	**1.8**	**6.6**	**895**

∗There is no information about this parameter on the reference

From Table 5.4 the presented design is the second-fastest design and the second most consuming design. It is expected that at higher speeds VLSI designs will have a higher power demand. The results in [12] are the closest that could be found to the character-

istics of the technology employed. Focusing only on this data, the design made on this work has a higher memory cell count, has better access time, and lower dynamic power consumption. These three elements show that the design made is a competitive one for the given technology.

Chapter 6

Conclusions

This documented has presented the development of a semi-custom SRAM IP, with the corresponding integration to a typical digital automatic flow. A custom library has also been designed and properly characterized for this purpose, with a custom black box flow for its use in physical synthesis with other libraries. Finally, a mixed-signal environment is configured to have accurate simulations of complex analog designs with digital input signals and a digital verification.

The final design of the SRAM shows great potential, with a high density, fast and low consuming base design. The resulting characteristics of the design not only are positive, but it is also clear that there is still room for improvement.

The custom library methodology enables a whole new type of projects to be developed at the DCILAB. This workflow enables the creation of designs without the requirement of external libraries provided by vendors, considerably reducing the possible monetary cost of projects. Another positive aspect of the custom libraries is that they are fully compatible with the current synthesis environment and work flow.

The mixed-signal flow presented is also pushing forward the capabilities and coverage of work and investigation that the lab can handle. All of this project has its efforts aiming towards a complete custom environment, in which analog designs will require testing and verification. This is why having a reliable and quick methodology for simulation is necessary, just as the method shown in this work. It should be noted that analog simulation still present a higher detail than the mixed-signal flow. As seen in Chapter 5, the mixed-signal simulation did not detect the reading error seen in the analog simulation.

Although the synthesis of different SRAM arrays was not made, it has been verified that the black box approach allows the integration of SRAM blocks for physical synthesis. Increasing the array size would only be a matter of adding more memory instances to the starting Verilog file. This allows not only larger memories, but enables the possibility to change the memory behaviour through additional control logic in order to create, for example, shift register memories.

6.1 Future work and recommendations

Future work may involve three main issues:

First, redesigning the control unit. This circuit requires multiple iterations and analysis between its schematic phase and post layout phase to properly adjust the custom SRAM designs results. Second, reduce the size of the memory and use a simpler reading method to ensure proper characterization of the control unit. Third, the cells used for the custom libraries can also be improved to have a design that is more route friendly.

An additional update to the methodology presented for mixed-signal simulation is the use of FineSim VCS. This tool has all of the elements shown for FineSim but also enables the programming of digital input signals and verification techniques through Verilog files, allowing for enven more robust tests and easier verification techniques.

The following recommendations can be mentioned:

- Analog designs should always include corner analysis to ensure the functionality of the design in all the possible operational cases.
- Before a custom library is made, extensive documentation needs to be written. This documentation should define elements such as pitch, unit tile size and naming conventions for ports and cells, and a list of expected designs that the library should contain. This will minimize mistakes, especially if multiple people are working on the same project.
- Do not use global net names for PG ports when designing custom cells. This makes easier the library creation and synthesis process.
- Have a shared folder in the server that contains the manuals for the tools available for the students in the DCILAB.

Bibliography

[1] N. E. H. Weste and D. M. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*. 4th ed., 2013.

[2] Q. Li and T. Kim, "Analysis of SRAM hierarchical bitlines for optimal performance and variation tolerance," *2011 International SoC Design Conference*, pp. 412–415, 2011.

[3] B. Wicht, T. Nirschl, and D. Schmitt-Landsiedel, "A yield-optimized latch-ape sram sense amplifier," pp. 409 – 412, 10 2003.

[4] F. Herrero, "Diseño de circuitos de columna para memoria SRAM para su integración en un microprocesador con arquitectura RISCV," *Tesis de Licenciatura, Escuela de Ingienería en Electrónica, Tecnológico de Costa Rica, Cartago*, 2018.

[5] F. Herrero, "Diseño de amplificador de sensado de tensión para una memoria SRAM de 64x32x2," *Not published*, 2019.

[6] B. E. Rodríguez, "Desarrollo de un módulo de decodificación de la fila para una memoria SRAM de 2 kb, en un proceso CMOS de 180 nm," *Tesis de Licenciatura, Escuela de Ingienería en Electrónica, Tecnológico de Costa Rica, Cartago*, 2018.

[7] Synopsys, "SiliconSmart User Guide," 2019.

[8] J. Samandari-Rad, M. Guthaus, and R. Hughey, "VAR-TX: A variability-aware SRAM model for predicting the optimum architecture to achieve minimum access-time for yield enhancement in nano-scaled CMOS," *Proceedings - International Symposium on Quality Electronic Design, ISQED*, pp. 506–515, 2012.

[9] W. H. Du, M. H. Chang, H. Y. Yang, and W. Hwang, "An energy-efficient 10T SRAM-based FIFO memory operating in near-/sub-threshold regions," *International System on Chip Conference*, no. Mi C, pp. 19–23, 2011.

[10] Q. Shang, Y. Fan, W. Shen, S. Shen, and X. Zeng, "Single-port SRAM-based transpose memory with diagonal data mapping for large size 2-D DCT/IDCT," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 22, no. 11, pp. 2422–2426, 2014.

[11] S. Ataei and J. E. Stine, "A 64 kB Approximate SRAM Architecture for Low-Power Video Applications," *IEEE Embedded Systems Letters*, vol. 10, no. 1, pp. 10–13, 2018.

[12] K. Verma, S. K. Jaiswal, D. Jain, and V. Maurya, "Design and analysis of 1-Kb 6T SRAM using different architecture," *Proceedings - 4th International Conference on Computational Intelligence and Communication Networks, CICN 2012*, pp. 450–454, 2012.

[13] A. Agal and B. Krishan, "6T SRAM Cell: Design And Analysis," *International Journal of Engineering Research and Applications*, vol. 4, no. 3, pp. 574–577, 2014.

[14] M. Kumar and J. S. Ubhi, "Performance evaluation of 6T, 7T & 8T SRAM at 180 nm technology," *8th International Conference on Computing, Communications and Networking Technologies, ICCCNT 2017*, pp. 3–8, 2017.

[15] M. Manimaraboopathy, S. Sivasaravanababu, S. Sebastinsuresh, and A. Rajiv, "Column decoder using PTL for memory," *IOSR Journal of Electronics and Communication Engineering*, vol. 5, no. 4, pp. 7–14, 2013.

[16] A. Azizi-mazreah, M. T. M. Shalmani, H. Barati, A. Barati, and B. R. Operation, "Delay and Energy Consumption Analysis of Conventional SRAM.," *International Journal of Electrical and Computer Engineering*, vol. 2, no. 1, pp. 35–39, 2008.

[17] R. J. Baker, *CMOS Circuit Design, Layout, and Simulation*. 3rd ed., 2010.

[18] S. A. D. V. Jacinto, A. J. M. Nanoz, J. R. A. Punzalan, F. A. Malabanan, A. S. Santos, J. N. T. Tabing, and S. M. Gevana, "Development of Low Power Full-Custom 1 Kb 8T Synchronous SRAM for Wireless Sensor Network in 90nm CMOS Process Technology," *IEEE Region 10 Annual International Conference, Proceedings/TENCON*, vol. 2018-October, no. October, pp. 2366–2371, 2019.

[19] F. Herrero, "Multiplexor de Columna para SRAM," *Not published*, 2019.

[20] B. S. Varela, "Diseño de un decodicador de fila parametrizable para memorias SRAM, tecnología CMOS de 180 nm," *Tesis de Licenciatura, Escuela de Ingienería en Electrónica, Tecnológico de Costa Rica, Cartago*, 2019.

[21] J. A. Junier, "Diseño e implementación de un controlador síncrono para SRAM en tecnología CMOS de 180nm," *Tesis de Licenciatura, Escuela de Ingienería en Electrónica, Tecnológico de Costa Rica, Cartago*, 2019.

[22] A. Stillmaker and B. Baas, "Scaling equations for the accurate prediction of CMOS device performance from 180 nm to 7 nm," *Integration*, vol. 58, pp. 74–81, 2017.

[23] W. B. Yang, C. H. Wang, I. T. Chuo, and H. H. Hsu, "A 300 mV 10 MHz 4 kb 10T subthreshold SRAM for ultralow-power application," *ISPACS 2012 - IEEE International Symposium on Intelligent Signal Processing and Communications Systems*, no. Ispacs, pp. 604–608, 2012.

[24] S. Miyamoto and N. Kobayashi, "Development of high-stability, low-leakage 6Tr-SRAM with single data line and single power supply using SOTB Process," *Proceedings of IEEE Computer Society Annual Symposium on VLSI, ISVLSI*, vol. 2018-July, no. 1, pp. 387–392, 2018.

[25] R. Fragasse, B. Dupaix, R. Tantawy, T. James, and W. Khalil, "Sense amplifier offset cancellation and replica timing calibration for high-speed SRAMs," *9th IEEE Latin American Symposium on Circuits and Systems, LASCAS 2018 - Proceedings*, pp. 1–5, 2018.