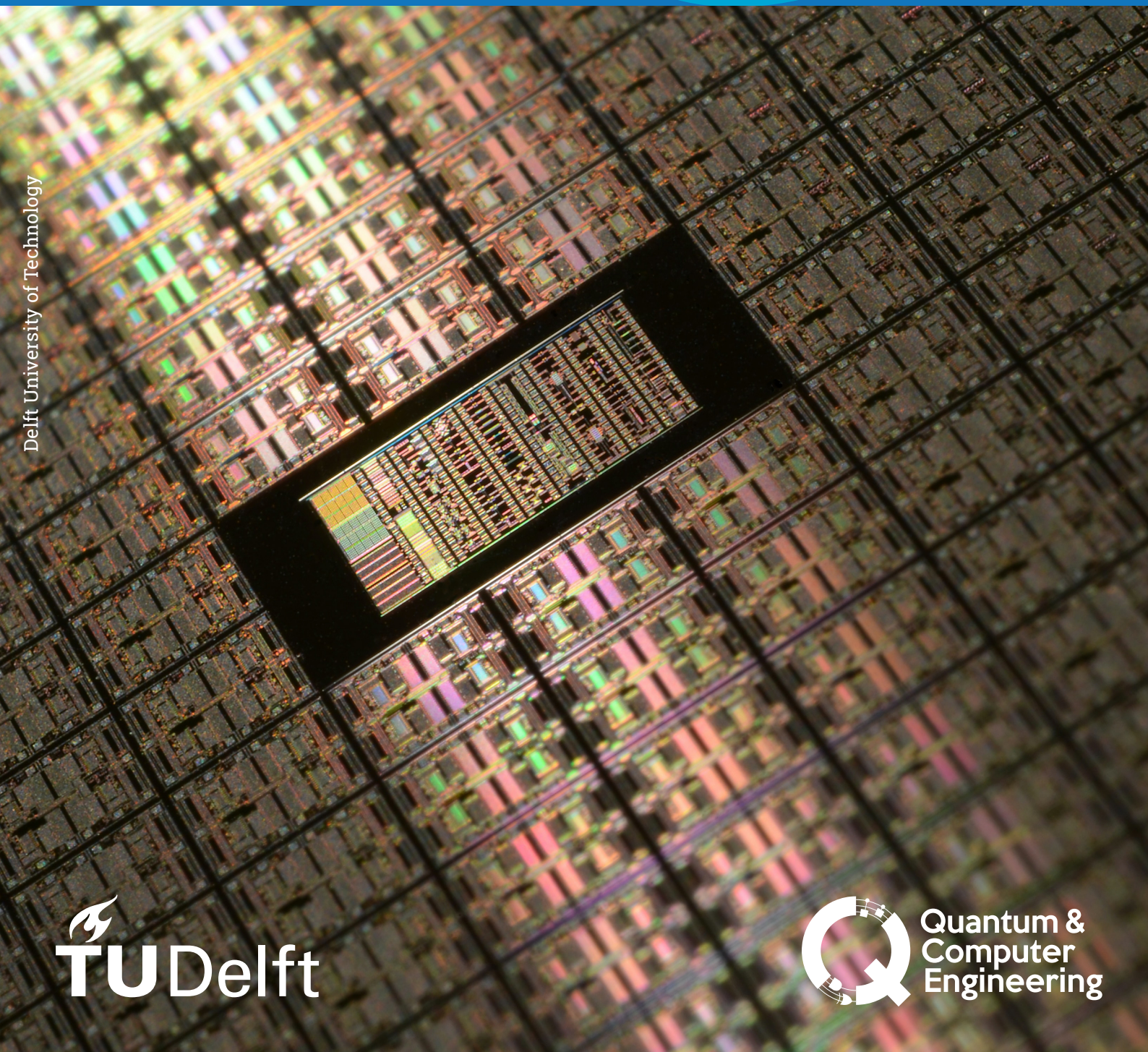
A FeFET based Reconfigurable Computing Architecture

Master Thesis

Quantum and Computer Engineering EWI
Jamie Teunissen



Delft University of Technology

TU Delft

Quantum & Computer Engineering

Preface

This thesis marks the end of my academic journey at TU Delft and the end of a challenging, exhilarating, and immensely rewarding chapter of my life.

Completing this project and reaching graduation was no small feat, and it was made possible by the unwavering support and guidance of several individuals. I extend my deepest gratitude to Stephan Wong for his consistent mentorship throughout this project. Your invaluable advice and steadfast support were instrumental not only in shaping my academic growth but also in improving the quality of this research. My heartfelt thanks also go to the members of the Q&CE group who generously shared their expertise and answered my queries.

Finally, I am deeply grateful to my family for their unending love and support. And to the wonderful friends and fellow students I have had the privilege to meet during my time at TU Delft and before. Your support has been a source of strength and inspiration throughout this journey.

Jamie Teunissen
Delft, November 2023

A FeFET based Reconfigurable Computing Architecture

Master Thesis

by

Jamie Teunissen

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on 16 November 2023.

Thesis Committee	
Supervisor	Dr. ir. Stephan Wong
Thesis Committee Member	Dr. Chang Gao

Project Duration: December, 2022 - October, 2023
Faculty: Faculty of Electrical Engineering, Mathematics and Computer Science, Delft

Cover: Laura Ockel via Unsplash
Style: TU Delft Report Style, with modifications by Daan Zwaneveld

TU Delft

Abstract

This thesis delves into reconfigurable computing architecture (RCA) design based on emerging embedded non-volatile memory (eNVM) technologies. Traditional Field-Programmable Gate Arrays, while efficient, face inherent limitations when it comes to power consumption and reconfigurability, primarily due to their set amount of resources and reliance on SRAM cells. Emerging eNVM technologies, such as STT-MTJ, ReRAM, and FeFET, offer potential alternatives, each with unique benefits and drawbacks. In this research, we introduce a novel RCA architecture built upon FeFET eNVM technology, which we called the 'Unified Tile'. This architecture is distinctive for its 2D layout of homogeneous tiles, these tiles integrate the three primary functions of an RCA: memory, interconnect, and logic. Our proposed design emphasizes the potential advantages of eNVM technologies over conventional SRAM-based FPGA components.

To validate our design, we employ an event-based simulator developed on the SystemC framework. Through evaluations and benchmarks, our 'Unified Tile' demonstrated notable improvements in power, area, and delay metrics compared to traditional FPGAs. Moreover, when compared with other state-of-the-art eNVM-based RCAs, our architecture showcased competitive, performance. These findings underscore the potential of eNVM-based RCAs as a viable alternative to SRAM-based FPGAs. This research shows the potential of eNVM technologies in reconfigurable computing architectures, with an emphasis on a FeFET-based design. It has also highlighted areas of concern, especially in programming latency, for these emerging technologies.

Contents

List of Figures

List of Tables

Nomenclature

Abbreviations

Abbreviation	Definition
AI	Artificial Intelligence
ASIC	Application Specific Integrated Circuits
BL	Bit-Line
BRAM	Block Random Access Memory
BSIM	Berkeley Short-Channel IGFET Model
CB	Connection Block
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
EDP	Energy-Delay Product
eNVM	Embedded Non-Volatile Memory
EMB	Embedded Memory Block
FeFET	Ferroelectric Field-Effect Transistor
FPGA	Field-Programmable Gate Arrays
HDL	Hardware Description Language
I/O	Input & Output
LB	Logic Block
LUT	Look-Up Table
ML	Machine Learning
MOSFET	Metal-Oxide Field-Effect Transistor
MTJ	Magnetic Tunnel Junction
MUX	Multiplexer
NMOS	N-channel Metal-Oxide-Semiconductor
PDP	Power-Delay Product
PMOS	P-channel Metal-Oxide-Semiconductor
RCA	Reconfigurable Computing Architecture
ReRAM	Resistive Random-Access Memory
SA	Sense Amplifier
SB	Switch Block
SRAM	Static Random-Access Memory
STT-MRAM	Spin Transfer Torque Magnetoresistive Random-Access Memory
TCAM	Ternary Content-Addressable Memory
WL	Word-Line

1

Introduction

1.1. Motivation

Field-programmable gate arrays (FPGAs) are the most common example of fine-grain reconfigurable computing architectures. FPGAs allow hardware designers to (re)configure the datapaths implemented on the chip after production. This fundamentally differs from conventional computing systems like PCs or mobile phones, where programs are composed of sequentially performed instructions. The advantage of FPGAs is that changing the ”hardware” for the needs of an application can significantly increase the efficiency of that application.

FPGAs comprise a programmable logic fabric with configurable logic and interconnect resources and are often supplemented with memory blocks. These components combined enable the reconfigurable capabilities of the FPGA. However, the topology of these resources can not be changed. In other words, we can not add more memory blocks or configurable logic after the production of the FPGA.

This constraint of FPGAs influences their efficiency and performance. For instance, the physical placement of memory blocks within the FPGA fabric can affect the routing resources required for a given design. Data must be routed to the specific location of the memory blocks on the chip, which can increase the routing resources needed, potentially leading to higher delays and power consumption. These problems can especially arise when FPGAs are used for data-intensive applications, which are more common with the increased use of AI and other ML applications in recent years.

Liquid Silicon and Merging Everything, two state-of-the-art reconfigurable computing designs, offer an interesting solution to overcome the limitations of current FPGA architectures. They implement a homogeneous tile-based design based on Resistive RAM (ReRAM).

These architectures use a fabric of homogenous components that can be configured to serve as memory, logic, or interconnect resources. This allows designers to allocate resources according to specific application requirements. Furthermore, using the ReRAM memory technology provides additional benefits, including high-density memory integration, data retention even when switched off, and lower leakage power.

ReRAMs are a new memory technology from a category called emerging non-volatile memory (eNVM) technologies. Other memory technologies from this group, such as ferroelectric FETs (FeFETs), may offer significant advantages over ReRAM. One key advantage of FeFETs is their ability to be completely turned off, resulting in negligible leakage currents. In contrast, ReRAM devices exhibit leakage currents even when turned off. Another benefit is that FeFETs have separate read and write paths, unlike ReRAM, which are two-terminal devices. Finally, FeFETs require less power to program than ReRAM devices. These advantages make FeFET technology a promising candidate for this kind of reconfigurable computing architecture.

The use of the FeFET technology still needs to be thoroughly investigated, and this thesis will explore this technology and its possibilities to be used in reconfigurable computing architectures.

1.2. Research Objectives

In this research, we will explore how eNVMs like FeFET technology can be utilized to design a non-volatile, reconfigurable computing architecture that overcomes the limitations of current FPGA architectures and achieves higher performance and energy efficiency. Therefore we are going to compare the performance characteristics of emerging eNVM technologies, including FeFETs, ReRAM, and STT-MRAM, within the context of a homogeneous tile-based design for reconfigurable computing. Therefore, our main research question can be formulated as follows: **How can we create a reconfigurable hardware structure utilizing emerging eNVM technology that is capable of flexibly and interchangeably operating as functional, routing, and memory structures?**

This thesis contains the following research objectives:

1. Conduct a thorough examination of the state-of-the-art in non-volatile memory technologies, including FeFET, ReRAM, and STT-MRAM, in order to comprehend their current capabilities and limits within the context of reconfigurable computing.
2. Design a hardware architecture based on those emerging non-volatile memory technology for reconfigurable computing.
3. Specify electrical models for the different components used in the different parts in the hardware design.
4. Conduct a functional verification of the proposed design.
5. Compare the performance and power consumption.

1.3. Methodology

We adopted a systematic approach to our research methodology. The first step involved conducting a thorough literature review to gain insight into the current state-of-the-art of eNVM-based reconfigurable computer architectures. Building on this knowledge, we proposed a novel fine-grain Reconfigurable Computer Architecture that utilizes the FeFET device as a basis for constructing its components. To evaluate the efficacy of our proposed architecture, we conducted a functional verification and an assessment of its electrical performance using an event-based simulator. As a basis for the simulator's parameters, we use a combination of SPICE circuit-level simulations and characteristics obtained from literature. We used a comparative analysis based on size, energy dissipation and latency to benchmark the results of our novel architecture against state-of-the-art architectures based on emerging eNVM technology and against the common RCA Platform, the FPGA. Finally, we compared the performance of different memory technologies when applied to our architecture.

1.4. Thesis Organization

We introduce the primary research topic and its significance in this Chapter. Subsequent Chapter 2 provides essential background information, including concepts related to FPGAs, eNVM technologies, and power dissipation in CMOS circuits.
Following the background information we present a literature review in chapter 3. A comprehensive examination of reconfigurable computer architectures based on eNVM technologies. This review also assesses the current state of eNVM technology.
Our design for a novel RCA based on FeFET technology is presented in Chapter 4. This section discusses the design objectives, architectural components, and electrical implementation.
The subsequent Chapter 5, Method of Verification and Testing, explains the verification techniques applied to the previously mentioned architecture. We also present our comparison methodology and benchmarks that aim to validate and test our proposed architecture.
Chapter 6 presents the findings from simulations, including dynamic energy consumption, verification, and electrical performance analysis. And Chapter 7 gives a summary of the thesis, and we discuss the contributions of this work and give recommendations for future work.

2

Background

This chapter provides essential background information for the topics covered in this thesis. We begin by discussing Field-Programmable Gate Arrays in Section 2.1. An FPGA is the most common implementation of a fine-grain reconfigurable computing platform. Understanding FPGAs helps in grasping the components and structures of architectures designed for reconfigurability. We then move on to Embedded Non-Volatile Memory (eNVM) technologies in Section 2.2, presenting an overview of the different types available. We will highlight the benefits of emerging memory technologies like FeFET and ReRAM compared to a traditional memory technology; SRAM. The chapter concludes with an brief refresher to the basic concepts related to power dissipation in CMOS circuits in Section 2.3.

2.1. Field-Programmable Gate Arrays

Reconfigurable computing systems are computer architectures capable of reconfiguring their ”hardware” to implement specific tasks or operations. Distinct from conventional fixed-architecture systems, these reconfigurable platforms can reconfigure their data-paths after fabrication. This often translates to improved efficiency and execution speed in comparison with fixed-architectures.
This section aims to introduce the fundamental components of FPGAs and their interactions. A comprehensive understanding of these elements is essential for recognizing the architectural requirements of a FPGA-like system.

2.1.1. Architectural overview

Boutras' work, as referenced in [1], provides an comprehensive overview of FPGA components. At its core, an FPGA is structured around programmable Input/Output (IO) interfaces, configurable interconnections, and configurable logic.

- **IO interfaces**: These gateways control communication between the FPGA and its external environment. These interfaces are responsible for the bidirectional exchange of data and control signals.
- **Configurable interconnections**: Crucial for signal routing among FPGA's logic blocks, these blocks form the backbone for all datapaths.
- **Configurable Logic**: Foundational to the FPGA is the configurable logic and can implement any logic function. Offering the inherent reconfigurability FPGAs are used for.

A notable feature in many modern FPGAs is the inclusion of Embedded Memory Blocks (EMBs). They are integrated within the FPGA architecture to accommodate data storage requirements for various applications. This not only improve the FPGA's computational capabilities but also optimizes data access times, further tailoring the FPGA's performance to application-specific needs.

Island-style Architecture

While various architectures exist for FPGAs, we will focus on the island-style FPGA to illustrate some core concepts and functional aspects. Figure 2.1 illustrates the island-style FPGA consisting of Config-

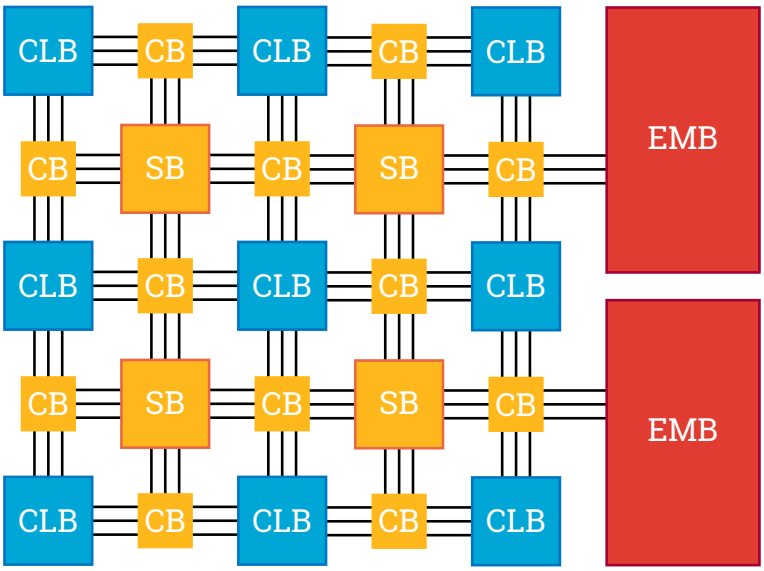


Figure 2.1: Simplified visualisation of a Island-based FPGA

urable Logic Blocks (CLBs) as its primary computational units. These CLBs, IO and EMBs are interconnected by Connection Blocks (CBs), which route the internal signals within the FPGA. The Switch Block or Interconnect Block (SB) acts as a higher routing entity, governing the signal pathways between CBs. It is important to highlight the significance of routing in FPGAs. As discussed by [1], the routing accounts for approximately 50% of the FPGA's total area, exerting a substantial influence on the critical path delay. The efficiency of the routing is crucial because it directly influences the chip's overall performance. FPGAs utilize a hierarchical routing structure. At the basic level within a CLB, there are intra-connections. These connections are localized, primarily facilitating communication within the same CLB. On a broader scale, SB comes into play when interactions between different CBs and they connected components are required.

2.1.2. Design of Components

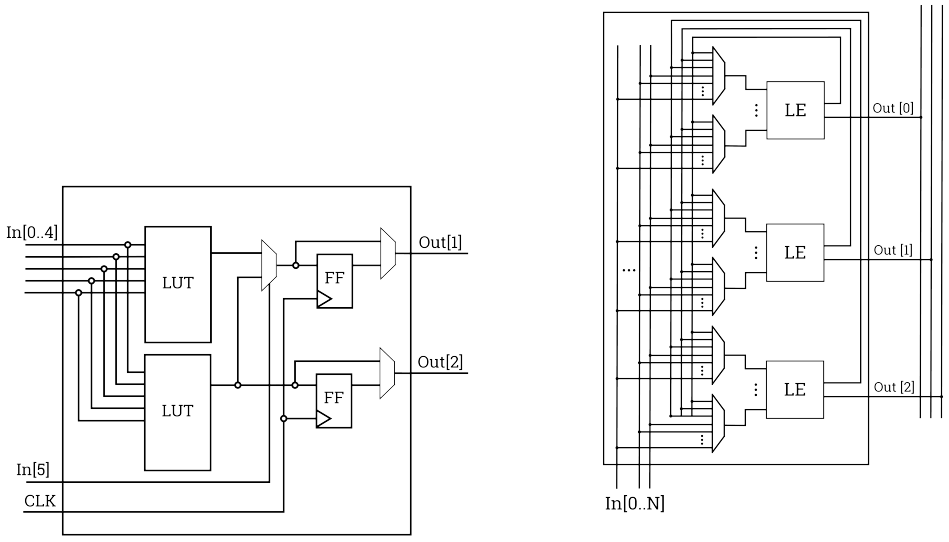
FPGA architecture necessitates the retention of configuration data for the reconfigurable elements. This information stored is commonly stored in Static RAM (SRAM). The subsequent sections will further discuss the components which can be reconfigured for this purpose we give special attention to the use of the configuration data.

Configurable Logic

At the heart of the FPGA's adaptive capabilities lies its Configurable Logic Blocks (CLBs). These units are specifically designed to implement any user-defined boolean functions.
Each CLB consists of numerous logic elements. These are interconnected through reconfigurable internal routing, thereby providing a means to internally define the data paths inside of CLB. Figure 2.2b gives the internal structure of a CLB, note that an CLB consists of multiple Logic Elements (LEs).
Delving deeper, the logic elements encapsulated within each CLB are the functional pillars that execute specific logic operations. Two key components within these logic elements are:

- **Lookup Tables**: These are essential for representing and executing complex Boolean functions. Their strength lies in their ability to offer a flexible and configurable approach to modeling logic behavior, allowing designers to map any arbitrary Boolean function to them.
- **Flip-Flops**: Acting as memory elements, Flip-Flops store and synchronize data within the CLB. They are not just mere storage units; their capability to retain state information is instrumental in enabling sequential logic operations, which are needed for numerous applications.

Combining LUTs and Flip-Flops provides everything needed to implement combinatorial and sequential logic operations.



(a) Schematic of a Logic Element **(b)** Schematic of a Configurable Logic Block

Figure 2.2: Schematic of primary logic elements of a FPGA

Look-up Tables

At a fundamental level, a LUT is a memory component. Each address corresponds to a specific combination of input values, and the data stored at that address represents the resulting output of the Boolean function for those inputs. The memory is populated with the pre-calculated outcomes for each possible combination of inputs. The illustration in Figure 2.3a showcases the structural implementation of an SRAM-based LUT. This figure displays a structure that can select a specific SRAM cell. The SRAM cell contains the output appropriate for the given value of ”ABC”. Rather than performing the computation, the LUT ”looks up” the precomputed result. An N-bit LUT can represent and execute any N-bit Boolean function to store and recall outcomes for all 2^N possible input combinations.

Interconnetions

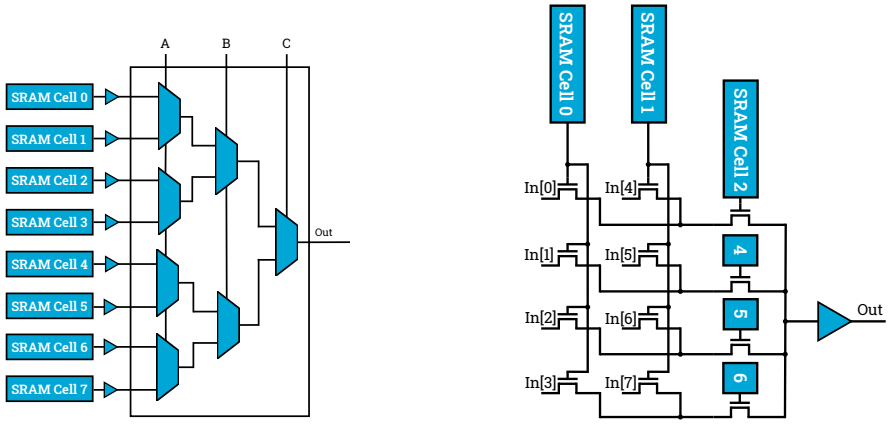
Building on the understanding of the island-style FPGA, the role of interconnections is integral in achieving the functionality of the FPGA as they define the datapaths. CBs serve as connecting components to internal signal paths.

Signals routed via Connection Blocks remain confined within their domain. This means that if a signal is on line 2, it stays on that line. A CB can not change the position of that signal to line 3. The SB is used to route signals to other domains, thus offering more complex interconnection functionalities. Modern FPGAs also implement a hierarchical routing structure, meaning signals can be routed over larger distances. SB also enables the switching over this hierarchy.

In Figure 2.3b we present an example of a routing resource. In this diagram, several pass transistors can be controlled like a switch. The configuration bits are used to ’open’ a specific switch, giving precise control over the datapaths without physically changing the lines.

Role and Impact of SRAM Cells in FPGAs

SRAM cells serve as the primary means of saving the configuration to determine the datapaths and implemented logic. That is why they can be found in a lot of essential components like CLBs, CBs and SBs. Their use in the FPGA architectures comes at a cost. Despite their versatile role, SRAM cells significantly contribute to the FPGA’s static power consumption. Static power denotes the energy drain when the FPGA is idle, not processing any active tasks. Research, as highlighted by [2], reveals that SRAM cells can be responsible for nearly 38% of an FPGA’s total static power. The inherent nature of SRAM cells, which demand a continuous power source to retain data, results in this significant power expenditure even when the FPGA is not actively computing.



(a) Implementation of a LUT

(b) Routing Resources

Figure 2.3: Visualisation of several resource

2.1.3. Summary

FPGAs are a reconfigurable computing systems which enable designers to reconfigure hardware for the needs of applications. Central to their design are CLBs and LUTs, which are key in defining the FPGA's reconfigurability and implementation of logic. Interconnections support these logic elements by supporting configurable datapaths.

Embedded within these components are SRAM cells, which serve as the configuration data. While these cells grant the FPGA its reconfigurability, they also introduce challenges, particularly in static power consumption. The sophisticated topology and discussed components collectively show the general structure needed to build a fine-grain reconfigurable computing platform.

2.2. Embedded Memory Structures

Embedded memory devices are integral components dedicated to storing data within embedded systems. These devices, often used in specific systems for applications ranging from medical devices and consumer electronics to computing systems, necessitate compact, power-efficient, and fast memory solutions. With the increasing demand of applications to the capabilities of memory, a surge in the development of new memory technologies. Figure 2.4 provides a taxonomy of some relevant technologies to this thesis.

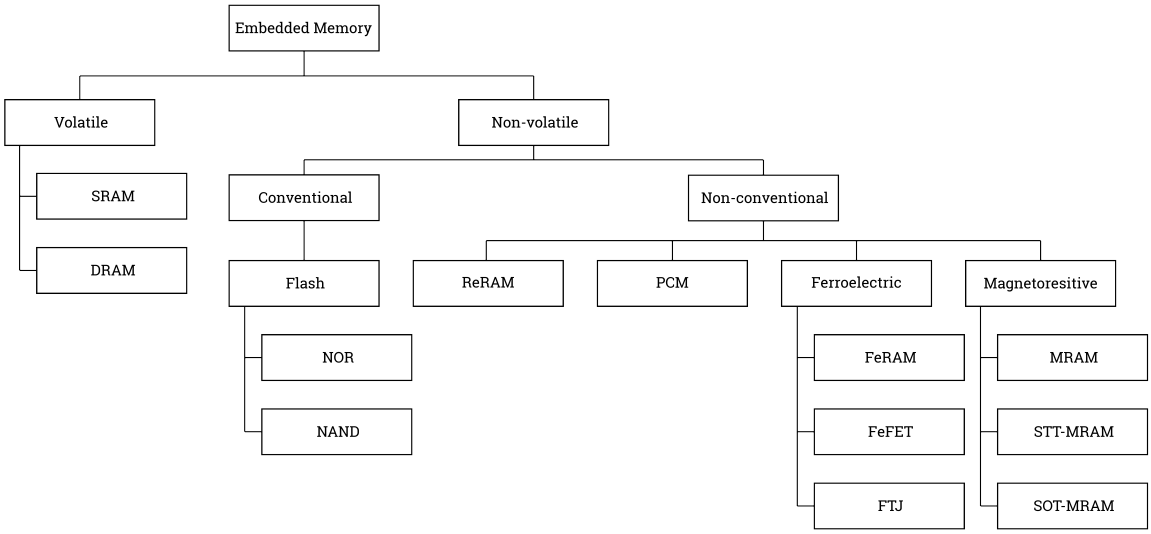


Figure 2.4: Taxonomy of embedded memory technologies based on [3]

2.2.1. SRAM

SRAM, or Static Random Access Memory, is a type of memory technology using CMOS (Complementary Metal-Oxide-Semiconductor). Its utilization is widespread due to its low switching delays and power efficiency relative to other memory types. One of the defining characteristics of SRAM cells is their dense structure, enabling them to be effectively integrated in crossbar array configurations, which allows the sharing of controlling and peripheral circuitry.

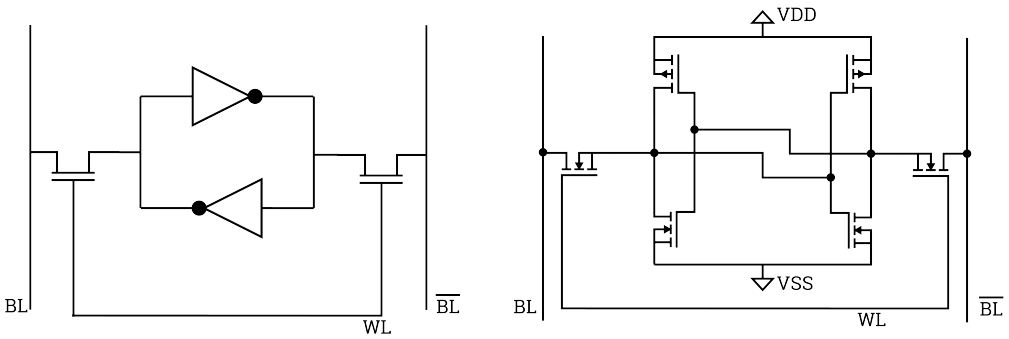


Figure 2.5: Schematic SRAM

Figure 2.5 illustrates a standard 6-transistor SRAM cell. At the heart of this cell, the core data storage mechanism is realized between two cross-coupled inverters. Each inverter comprises one PMOS (p-channel metal-oxide-semiconductor) transistor and one NMOS (n-channel metal-oxide-semiconductor) transistor. The combination of the two makes this circuit CMOS.
The bistable nature of the cross-coupled inverters enables the SRAM cell to retain its stored value.

As long as power is supplied to the cell and no external influences override the current state, the complementary voltage levels at the output of each inverter ensure that the stored data remains stable and unchanged. The information is effectively held in a feedback loop between the two inverters: when one outputs a high voltage level (logical '1'), the other outputs a low voltage level (logical '0'), and vice versa.
Reading data from an SRAM cell starts with a pre-charging phase. Here, the bitlines are initially set to a high voltage level, usually equating to the supply voltage. Following this pre-charge, the word-line activates, inducing a voltage drop in one of the bitlines while the counterpart remains unchanged. This voltage disparity becomes detectable by a sense amplifier, which converts the differential into a representative digital signal, indicating the data stored within the SRAM cell.
To program or write data into the SRAM cell, the bitlines, denoted as BL and $\overline{BL}$, which convey the data to be stored, are energized with an appropriate voltage. This power ensures that the voltage levels on the bitlines can supersede the current state of the SRAM cell, thus enabling the new data to be recorded. The controlled voltage levels dictate whether the cell should store a logical '1' or '0'. The new state effectively breaks the feedback loop between the two inverters, setting them to the desired state, after which the feedback mechanism takes over again to hold the newly written data stably.
SRAM cells are classified as volatile memory technology, requiring a continuous energy supply to retain the stored data. When the power supply to an SRAM cell is disconnected or interrupted, the data contained within the cell is lost. Unlike non-volatile memory technologies such as Flash memory, ReRAM or FeFETs, which can retain data even when power is removed, SRAM cells rely on the presence of a constant power source to maintain their stored information.

2.2.2. Introduction to Emerging eNVMs

Emerging embedded Non-Volatile Memories (eNVMs) are a interesting development in memory technology, offering unique advantages like non-volatility, which means that data is stored even if there is no voltage supplied. This section provides a detailed exposition on three prominent Emerging eNVM technologies: Resistive Random Access Memory (ReRAM), Ferroelectric Field Effect Transistors (FeFETs), and Spin Transfer Torque Magnetoresistive RAM (STT-MRAM).

ReRAM

Resistive Random Access Memory is an embedded memory technology that leverages the principle of resistance-based storage. At its core, a ReRAM device is characterized by two primary electrodes, between which resides a conductive filament. The inherent characteristics of this filament allow the ReRAM to act as a non-volatile memory, ensuring data retention without power. Several sources like [4, 5, 6, 7] give a good overview of the technology.

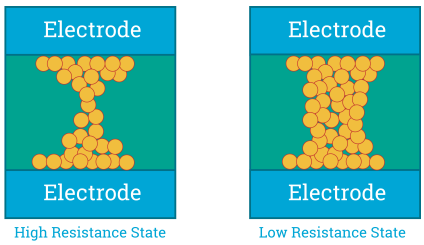


Figure 2.6: Filament forming in a ReRAM device

The fundamental property leveraged for data storage in a ReRAM device is the variable resistance of the conductive filament. The resistance can be modified by driving a specific current through the device, which in turn adjusts the composition of the filament. Depending on the specific technology variant, this filament manifest as vacancies in oxygen or as an aggregation of metal atoms. The various states of the filament, representing the different data states, are illustrated in Figure 2.6.
In operational terms, data in a ReRAM device is encoded in resistance states. These states can be determined using several sensing techniques. When define the current through the ReRAM cell as I_{sense} (Sense Current) during a sensing operation. Two current values, I_{HR} and I_{LR}, can be defined associated with the high-resistance and low-resistance states, respectively. By applying a differential voltage across the device terminals, one can measure variations in the I_{sense} to determine the stored data.
In terms of integration and architecture, ReRAM can like SRAM be integrated into high-density crossbar structures. However, to ensure individual cell accessibility within these arrays, a selection mechanism is unmissable. This is typically addressed using an access transistor paired with each ReRAM cell, leading to what is known as the 1T1R configuration. Figure 2.7a presents this configuration, emphasizing the paired transistor ensuring distinct cell selection during array operations.

(a) Schematic of a 1T1R cell

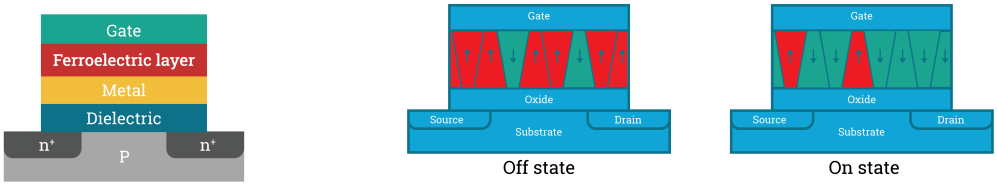
(b) Measuring a column of ReRAM cells

Figure 2.7: Common structures to incorporate ReRAM devices in crossbar arrays

In Figure 2.7b, a crossbar array with integrated ReRAM cells is illustrated. Through sensing of the current flowing through the ReRAM cells, it is possible to determine the number of cells in the ON state.

FeFET

Ferroelectric Field Effect Transistors (FeFETs) are a memory technology which stores information in the polarisation of a ferroelectric (Fe) layer. Ferroelectric materials possess unique properties wherein their electric dipoles can align in specific directions when subjected to an external electric field. This alignment or polarisation is retained even after removing the external electric field, providing a mechanism for non-volatile data storage. A discussion on the subject is provided in sources like [8, 9, 10]
As shown in Figure 2.8a, FeFETs slightly differ structurally from MOSFETs, as they feature an additional ferroelectric layer. FeFETs are highly compatible with CMOS architectures, given their structural resemblance to the more familiar MOSFET devices.
Data writing in a FeFET revolves around manipulating the polarisation state of its ferroelectric layer. The polarisation state of the ferroelectric layer can be changed by applying a specific gate voltage (V_G) to the device. Figure 2.8b visualises the polarisation states.
For reading stored data, while the intrinsic state is encoded in the polarisation of the ferroelectric layer, it translates as a change in resistance when a voltage differential is applied between the source and the drain and reading voltage is applied to the gate. The resulting current (I_{DS}) that flows in response to this voltage is determined by the polarisation state of the Fe layer. Thus, the intrinsic data representation in the polarisation state modulates the device's resistance, leading to observable current differences: I_{ON} or I_{OFF}.
However, one challenge associated with FeFETs, compared to certain other non-volatile memory types, is their endurance. Specifically, FeFETs typically exhibit a reduced number of read/write cycles before degradation, making their long-term reliability a consideration in specific applications.
Nonetheless, the tri-terminal design of FeFETs offers distinct advantages, such as operational flexibility, especially with the separation of reading and writing pathways.



(a) Gate stack of a FeFET device

(b) Visualisation of Ferroelectric switching behavior

Figure 2.8: FeFET Devices

STT-MRAM

Spin Transfer Torque Magnetoresistive RAM (STT-MRAM) emerges as a distinctive memory technology that encodes data through the mechanism of electron spin. This technology is discussed in the following sources [11, 12, 13] The device's foundational element is the Magnetic Tunnel Junction (MTJ),

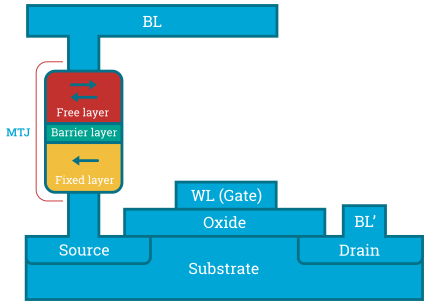


Figure 2.9: Visualisation of a Gate stack of STT-MRAM

which, in conjunction with a transistor, functions as the primary data storage unit. Figure 2.9 provides a visualization of the STT-MRAM, with emphasis on the MTJ structure.
Central to the function of the STT-MRAM is the MTJ, comprising two magnetic layers separated by an insulating barrier. Of these, the first retains a fixed magnetic direction, thereby earning its designation as the ”fixed” layer. The second, the ”free” layer, possesses a mutable magnetic orientation. This orientation can be modified by directing a current through the free layer, transferring some of the flowing electrons' angular momentum to it, leading to a change in its magnetic alignment.
A critical aspect of the STT-MRAM's operational principle lies in the relative alignment of these magnetic layers. When both layers possess matching magnetic orientations, the MTJ exhibits a Low-Resistance State. In contrast, a misalignment results in the High-Resistance State. This resistance variation, attributable to the layers' alignment, facilitates the encoding and reading of stored data.

Summary of Emerging eNVMs

eNVMs are a interesting developement in memory technology. For this thesis we consider three of these emerging eNVM technologies:

1. **Resistive Random Access Memory (ReRAM):** At its essence, the ReRAM utilizes resistance-based storage, primarily through the manipulation of a conductive filament situated between two electrodes.
2. **Ferroelectric Field Effect Transistors (FeFETs):** Distinguished by their storage mechanism, FeFETs rely on the polarization of a ferroelectric layer. By harnessing the unique attributes of ferroelectric materials. Their resemblance to MOSFET makes them compatible with CMOS.
3. **Spin Transfer Torque Magnetoresistive RAM (STT-MRAM):** This technology stands out with its data encoding mechanism rooted in electron spin. Central to its operation is the MTJ, which consists of two magnetic layers. The relative alignment of these layers' magnetic orientations—either matching or misaligned—dictates the resistance state of the device, forming the means for data storage.

it is evident that the landscape of memory technology is being progressively improved by the evolution of eNVMs, with each offering distinct operational principles and integrative capabilities.

2.3. Understating Power Consumption for CMOS

Power consumption is a significant factor that affects the cost of operating computing systems. In particular, the majority of computing systems use CMOS, and understanding the factors that contribute to power consumption in CMOS circuits is crucial. This includes both static and dynamic power contributions, as well as the importance of propagation delay. This section aims to explain the power consumption and delay of CMOS circuits.

Power Consumption of CMOS

The power consumption of CMOS is determined by two key attributes:

- **Complementary Design**: The combination of NMOS and PMOS transistors ensures that when one is ON (conducting), the other is OFF (non-conductive), virtually eliminating a DC path between the power supply and ground. This drastically reduces static power dissipation.
- **Capacitive Loading**: In CMOS circuits, a significant amount power is consumed during the charging and discharging of inherent parasitic capacitances. However, given the minimized DC path, this power consumption remains a fraction of what would be if a continuous direct path between the power supply and ground existed.

$$P = P_{static} + P_{dynamic} \tag{2.1}$$

In a CMOS, two main factors contribute to its overall power consumption. The first factor is static power consumption, which is the energy required to power the circuit. The second factor is dynamic power consumption, which is the power lost during the switching of components. You can see a visualization of this effect in Figure 2.10. The relation is given in Formula 2.1.

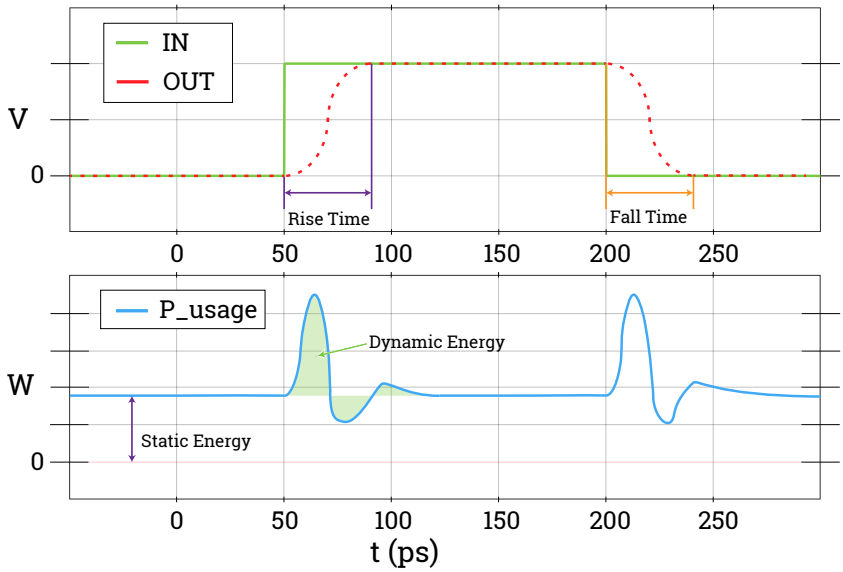


Figure 2.10: Example of graph of main components of sensing operation

Static Power

The static power consumption in CMOS circuits is caused by the the leakage current from drain to source. In figure 2.11 this is visualized by the red arrow. The leakage current is determined by an number of parameters of which one of the most apparent is the transistor scale. Decreasing the technology node will cause leakage currents to go up.
The static energy is modeled by the determining the leakage currents of the circuits and the voltage drop over the transistors. Which gives us the following formula.

$$P_{static} = (V_{DD})^2 I_{Leakage} \tag{2.2}$$

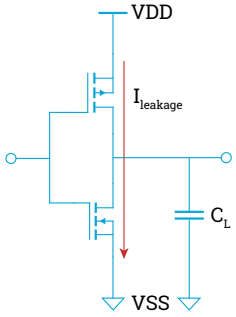


Figure 2.11: Static power dissipation of CMOS inverter

Dynamic Power

During the switching of CMOS device current will briefly flow between the two transistors. Because CMOS circuits are used to drive other MOS devices the dynamic power dissipation can be modeled by the input impedance of these circuits can be modeled by a capacitance. The relation between the current and the voltage for the switching of a single MOSFET is given in 2.3.

$$i_L = C_L \frac{\mathrm{d}v_o}{\mathrm{dt}} \tag{2.3}$$

The energy dissipated in the PMOS device during the switching from low to high is given in 2.4.

$$E_P = C_L V_{DD} \int_0^{V_{DD}} \mathrm{d}v_o - C_L \int_0^{V_{DD}} v_o \mathrm{d}v_o = \frac{1}{2} C_L (V_{DD})^2 \tag{2.4}$$

The energy dissipated in the NMOS device is the energy stored in the load capacitance which discharges over the NMOS. Therefore it yields the same formula as for the PMOS device thus $E_N = \frac{1}{2} C_L (V_{DD})^2$. The total energy disipated during a switching cycle is determined by the formula given in 2.5

$$E_T = E_P + E_N = C_L (V_{DD})^2 \tag{2.5}$$

If the inverter is switched with frequency f, then the total power dissipated over such a inverter is defined by formula 2.6. Where α is the activity factor defined by the amount of switching which occurs.

$$P_{dynamic} = \alpha f E_T = \alpha f C_L (V_{DD})^2 \tag{2.6}$$

Formula 2.6 shows a clear relation between the frequency and the chosen VDD. increasing either of the two would result in an higher power dissipation of the circuit.

(a) Switching from 0 to 1

(b) Switching from 1 to 0

Figure 2.12: Dynamic power dissipation of CMOS inverter

Propagation Delay
There are multiple levels at which an delay from input to output is being produced. The main contributing factor to this delay is the switching of MOS devices used in the tile. This switching time is ultimately determined by the amount of load gates an driving CMOS structures has to charge. In figure 2.12 these load gates are modeled as an capacitance. The load capacitance is directly proportional to the amount of load gates which are placed at the output of, in this case, the inverter. The amount of load gates behind the driver circuit is often called the fanout.
Because the gate capacitance is directly proportional to the gate area of the load capacitance we observe the following relation between the propagation delay time the fanout and the gate area. This relation is given in formula 2.7 where N is the fanout W is the gate width and L is gate length and t is the propagation delay time.

$$t \propto \frac{N\,(W \cdot L)_L}{\left(\frac{W}{L}\right)_D} \tag{2.7}$$

2.4. Conclusions

In this section, we provided essential background information on the different subjects related to this thesis. To start, we introduced the FPGA, which is a fine-grain RCA. We explained how this computer architecture allows hardware designers to implement any circuit design on this chip. The FPGA has four key functionalities that implement its reconfigurable capabilities: IO, Configurable Logic, Configurable Interconnections, and Memory. We also explained how these components are connected to form the complete architecture of the chip.
Moving forward, we delved deeper into memory technologies, starting with a discussion of SRAM. We explained how this technology holds the configuration data for different FPGA components. We explained how SRAM is constructed and how they work and explored a new group of memory technologies called emerging eNVM, which includes ReRAM, FeFET, and STT-MRAM. We discussed how these technologies function and what their benefits are over SRAM.
To conclude this chapter, we provided a refresher on the power dissipation of CMOS circuits. CMOS circuits are often used in computer hardware, and understanding the dynamics governing their main contributions to power consumption is key in analyzing these kinds of circuits.

3

Literature Review

Reconfigurable computer architectures are continually evolving, eNVM systems present promising potential to further reshape and inspire new design paradigms in this domain. This literature review concentrates on the potential integration and implications of eNVM within the context of reconfigurable architectures.
This chapter is split-up into two analyses:

- **State-of-the-Art analysis emerging eNVM:** An examination of eNVM technologies, centered on their electrical characteristics.
- **State-of-the-art eNVM based RCA architectures:** An analysis of reconfigurable computing architectures built on eNVM, comparing two distinct architectures to discuss their relative strengths and weaknesses.

This review will first present an analysis of the latest advancements in eNVM technology in Section 3.1. After which in Section 3.2 we will discuss their integration into reconfigurable architectures. Where an evaluation with the current state-of-the-art architectures is presented. After which we provide a brief discussion on the use of eNVM technology as a substitution for SRAM based components in Section 3.3. In Section 3.4 we present the conclusions of this chapter.

3.1. State-of-the-Art analysis emerging eNVM

Following the exposition of relevant memory technologies presented in Section 2.2, this section extends the discussion by examining the performance of state-of-the-art eNVM technologies. Initially, we will present the current research landscape, and after that, a more detailed exploration of the electrical characteristics of each memory technology is given. The primary objective is to understand the technical attributes of each technology and position them within the context of the current state of research.

3.1.1. Current state of research

To understand the research direction of emerging eNVM technologies, we focus on performance metrics, particularly those benchmarked against the established SRAM technology. Figure 3.1, taken from [14], shows a dominant research interest in the STT-MRAM and the ReRAM technology.
Significant efforts have been focused on improving the performance of eNVM technologies to align with SRAM specifications, which commonly serve as the last-level cache (LLC) in the memory hierarchy of computing systems. However, it is essential to note that despite progress being made, eNVM technologies are yet to reach the same level of performance as SRAM. To compare the performance of these memory technologies across different levels in the memory hierarchy, it is necessary to examine each level's distinct purposes. Computing systems have multiple levels in their memory hierarchy, with permanent storage solutions such as hard drives or eFlash at the highest level (slowest) and cache/register storage at the lowest level (fastest), offering the fastest access to data. As indicated by [13], the distinct requirements imposed by different levels in the memory hierarchy highlight the need for careful selection of one of these technologies.

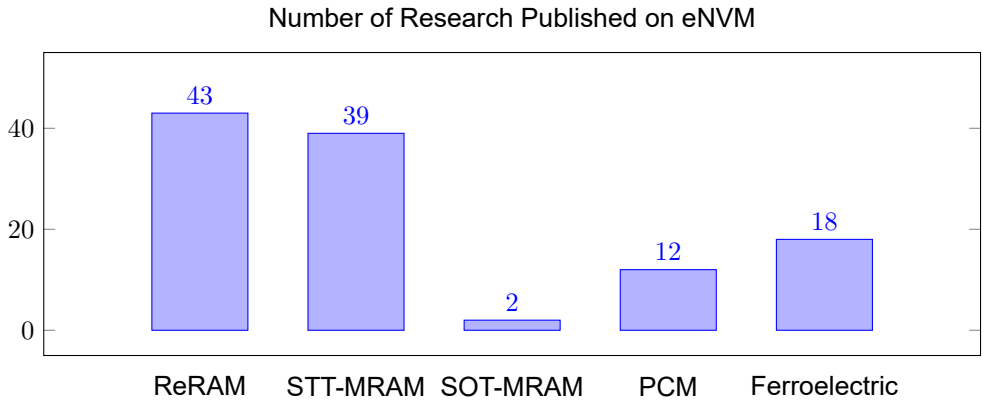


Figure 3.1: Bargraph of published work on subject

Table 3.1: Different merits of current state-of-the-art eNVM technolgies

	SRAM	DRAM	Flash	PCM	ReRAM	STT-MTJ	FeFET
Endurance	$> 10^{16}$	$> 10^{16}$	$< 10^{5}$	$10^{8} - 10^{11}$	$10^{6} - 10^{12}$	$> 10^{15}$	$> 10^{10}$
Retention	X	$> 64ms$	$> 10y$	$> 10y$	$> 10y$	$> 10y$	$> 10y$
On/Off Ratio	X	X	$> 10^{8}$	$> 10^{4}$	$10^{3} - 10^{7}$	> 2	$> 10^{8}$
Write Energy	fJ	$10fJ$	$10fJ$	$10pJ$	$100fJ$	$100fJ$	$10fJ$
Write Speed	$< 1ns$	$< 10ns$	$< 10ms$	$< 100ns$	$< 10ns$	$< 5ns$	$< 10ns$
Read Speed	$< 1ns$	$< 10ns$	$< 10\mu s$	$< 10ns$	$< 10ns$	$< 10ns$	$< 10ns$
Size	$> 100F^2$	$> 6F^2$	$> 4F^2$	$> 4 - 20F^2$	$> 4F^2$	$> 6 - 20F^2$	$> 10 - 30F^2$

Table 3.1, which compiles merits from recent studies [15, 7, 16], provides a basis for evaluating eNVMs relative to SRAM. None of the emerging eNVM technologies possess the same endurance, write and read speeds. However, they also indicate that these eNVMs can still be improved. One of those improvements is that size can still be reduced. While the current state of these technologies does not yet meet the requirements for LLC applications, they demonstrate improvements in energy efficiency and speed compared to traditional Flash memory. Most emerging eNVM technologies exhibit potential as alternatives to DRAM in terms of speed and energy usage. One of the most challenging merits of these technologies is their endurance.
Si, et al. [13] highlight the potential of emerging eNVM technologies. Despite their advancements, they are at this moment still unsuitable to replace SRAM cells in LLC applications at the current state of research. Another very prominent field of research for these emerging eNVMs are there integration in Processing-in-Memory (PiM).

Processing-in-Memory
PiM is an advanced architectural paradigm aimed at addressing the limitations of the von Neumann architecture, particularly the so-called ”von Neumann bottleneck”. This bottleneck pertains to the latency and energy overhead associated with moving data between the central processing unit (CPU) and main memory. PiM aims to alleviate this by incorporating processing capabilities directly within or close to the memory modules, enabling data processing tasks to be executed where the data resides.
Resistive memory technologies like ReRAM inherently supports in-memory computing operations, such as bitwise logic operations and matrix-vector multiplications. Several features of resistive memory technologies make it particularly amenable to PiM because this memory technology can perform weighted summations in an analog fashion. By reducing data movement, in-memory computations with ReRAM can lead to significant energy savings.

3.1.2. Comparison emerging eNVM
STT-MTJ exhibits promising results for memory applications among the emerging eNVM technologies. However, there are notable trade-offs compared with other technologies, such as ReRAM and FeFET. We will discuss the key trade-offs in this section.

On/Off Ratio
The on/off ratio, a metric representing the resistance difference between the on and off states, is a defining characteristic distinguishing these memory technologies. While STT-MTJ shows potential in various applications, its on/off ratio lags behind ReRAM and FeFET's. Though perhaps less consequential for pure memory applications, this disparity affects the scope of feasible use cases, especially when contrasted with FeFET, as elaborated in [17]. One implication of a lower on/off ratio is evident in particular SAs, which demand more significant area requirements to support the lower ratio. Another benefit of a high on/off ratio is that they can be placed in the path of signal as a switch. Because the high resistance can effectively block a signal from propagating further.

Device Access
As discussed in Section 2.2.2, ReRAM requires access transistors to achieve isolation from adjacent devices in a column due to their inability to gate or select a specific device. According to [18], the parallel operation of multiple ReRAM devices in a low-resistance state diminishes the overall resistance detected by the sense amplifier, which increases power consumption. To pinpoint the state of an individual device, incorporating an access transistor is needed to alleviate this problem to deactivate alternate paths fully. STT-MRAM and FeFET address this challenge since they are inherently three-terminal MOSFET device. This design allows them to segregate control and sensing paths, given that both necessitate a read voltage at their gates during sensing operations. If this read voltage is not applied the path is always closed.

Endurance
Endurance emerges as a important factor where FeFET appears less robust than its counterparts. Indications point towards STT-MRAM excelling in endurance, mirroring SRAM attributes closely. This positions STT-MRAM as a potential substitute, especially in applications demanding recurrent memory modifications, such as LLCs. However, as noted in [15], FeFET remains a contender, displaying attributes of high performance, energy efficiency, and density, especially for next-gen AI computing and data processors.

Other Applications
When employed as switches, the capabilities of these devices become critical. STT-MRAM, for instance, struggles to effectively cut off a path with sufficiently high resistance, as emphasized by [17]. Conversely, while ReRAM can sever a route, its lower resistance results in higher power consumption than a FeFET device.
ReRAM balances the discussed technologies, boasting a high on/off ratio and endurance. Additionally, the two-terminal configuration of the device allows it to function as a switch between two points even when not selected, presenting a feasible solution for applications like routing.

3.1.3. Conclusions
Exploring emerging eNVM technologies underscores the diversity in their characteristics and potential applications. While STT-MRAM is a promising candidate, especially in endurance, it is evident that no single technology completely outclasses SRAM. ReRAM's two-terminal configuration offers unique opportunities, especially in routing applications. Conversely, while FeFET has limitations in endurance, its performance in other areas cannot be disregarded. In essence, the selection of an eNVM technology remains contingent upon the specific needs of an application. Given the current state of research, these technologies are still very much in development and are still expected to improve in the future. This makes emerging eNVM technologies as an replacement for SRAM and other conventional technologies in reconfigurable computing systems still an interesting field of research.

3.2. State-of-the-art eNVM-based RCA architectures

Building upon our understanding of eNVM technologies and their electrical characteristics, this section delves into how they are integrated within state-of-the-art reconfigurable computing platforms. In a previous project [19], we already analysed the field of processing-in-memory and found some compelling use cases of the different technologies. This work will further discuss the most suitable designs as an RCA. Specifically, we highlight the innovative design approaches of these suitable RCA architectures that leverage eNVM.
Two architectures, *Liquid Silicon* [20] and *Merging Everything* [23], stand at the forefront of this development. While both capitalize on using a 2D grid of homogenous components as a design principle, their methods of implementing reconfigurability and underlying design philosophies are distinct.

The discussions that follow will:

1. Delve into the architectural specifics of *Liquid Silicon*, emphasising its homogeneous tile-based approach.
2. Examine the *Merging Everything* architecture, with its unique LUT-based strategy and the advantages this strategy gives for implementing complex boolean functions.
3. Presents a detailed comparison of the two, their merits, potential limitations, and the more significant implications for developing reconfigurable computing systems based on emerging eNVMs.

First, we will delve into the design of *Liquid Silicon*, offering insights into its architectural intricacies and how it implements an RCA based on emerging eNVM technology.

3.2.1. Liquid Silicon

Yue Zha and Jing Li [20] introduced Liquid Silicon, a novel reconfigurable computing architecture. They conceptualized a two-dimensional array of homogeneous ”tiles,” where each tile can switch between computation and storage. Additionally, to enable Verilog code synthesis for this distinct architecture, a CAD toolset was developed leveraging the VTR framework [21].

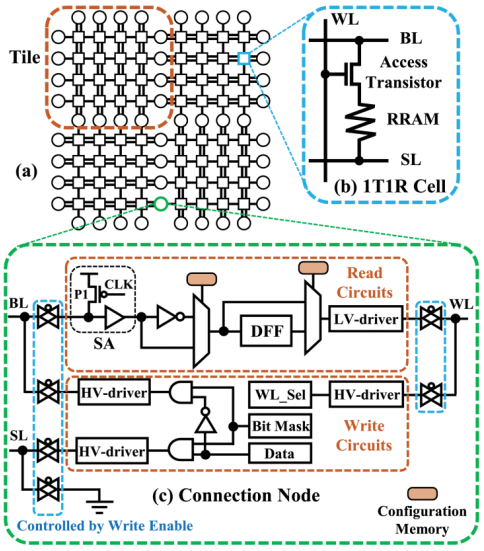


Figure 3.2: Overview of the architecture of Liquid Silicon from [20]

Homogeneous Tile

The architecture’s basis lies in a ReRAM crossbar array. Every row/column of this array is accessible via connection nodes, and adjacent tiles can interface with these nodes. In Figure 3.2, the tile is represented in red, and its connection node is in green. The connection node can function as either an input or an output, depending on its configuration. When set as an input, it drives the Word Line (WL), which controls the access transistor. The state of the ReRAM device is determined with an SA connected via the sensing line (SL) and can activate the connection node of an adjacent tile. Given the inherent flexibility of the tile-based design, each can operate in one of several modes, each serving a specific purpose.

Memory Mode

By selecting a single Word Line (WL) and reading all Bit Lines (BL), all states of the ReRAM devices in that row can be determined simultaneously. A neighbouring tile translates an N-bit address to the appropriate WL. A notable feature of this architecture, which comes with all homogenous architectures, is the flexibility in memory block placement within the fabric. The researchers claim that with this feature, routing paths are shortened by situating memory closer to its point of use.

Computation Mode
The computation mode leverages the inherent NAND operation enabled by the crossbar array. This mode allows the synthesis of complex combinatorial logic by sequencing NAND operations on the fabric. Given that any boolean function can be constructed using NAND gates, a wide variety of combinatorial logic functions can be mapped onto this architecture. Sequential circuits are realized by integrating a skippable flip-flop in the connection node.

Interconnect Mode
The interconnect mode capitalizes on the native behaviour of the NAND operation on the desired column of the crossbar array. This forms a connection, enabling a specific row to be linked with a chosen column. Such a mode enables signal routing within the architecture, permitting precise control over connections and data flow. The flexibility of this mode aids in optimizing the mapping of designs onto the Liquid Silicon fabric.

Search Mode (TCAM)
In this mode, tiles transform into embedded Ternary Content-Addressable Memory (TCAM) blocks. The unique design of the RRAM crossbar array offers direct TCAM functionalities. When introduced with a search key from neighbouring tiles, it is parallel to stored data entries. This concurrent comparison greatly benefits applications needing fast data retrieval, exemplified in networking or database search tasks. Upon finding a match, results are directed to adjacent tiles for subsequent processing.

Results
The findings of this research are notable. A remarkable 62% reduction in the overall area compared to SRAM FPGA was reported in [22]. Furthermore, compared with an FPGA design using ReRAM technology, the architecture still boasts a 13% size reduction. This highlights that the space efficiency is not merely due to the smaller footprint of ReRAM but also stems from the circuit-level design.

3.2.2. Merging Everything

Chen, et al. introduced Merging Everything [23], a novel reconfigurable computing architecture different from traditional designs. While it leverages ReRAM technology and presents a grid of homogeneous tiles like Liquid Silicon, its communication and operational modes approach differs.
A key difference with the Merging Everything design is that neighbouring tiles do not directly influence the Bit- and Wordlines of the crossbar array. Rather than interconnecting neighbouring tiles directly via these lines, this design uses more components in the periphery to enable address decoding and row selection.

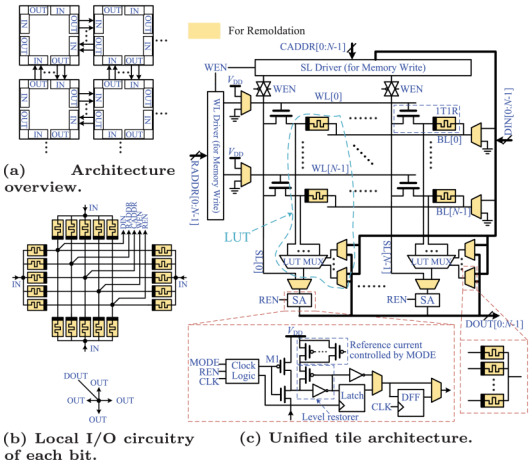


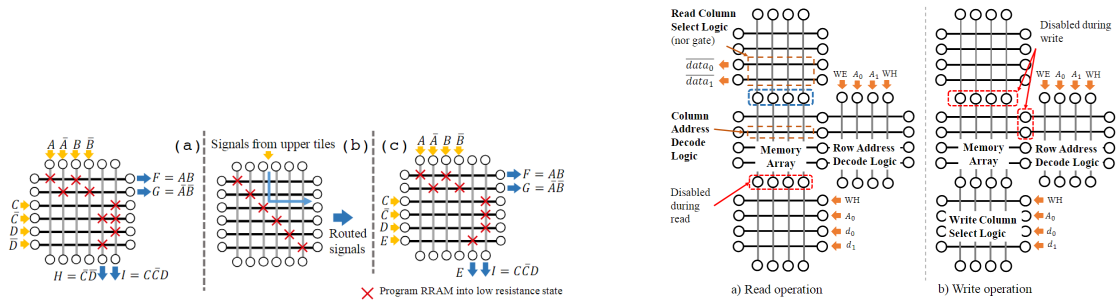
Figure 3.3: Overview of the architecture of Merging Everything from [23]

Operational modes
Like Liquid Silicon, Merging Everything also implements three specific modes for its homogenous tile component: Logic mode, Memory mode, and Interconnect mode. Another key difference in this design is the possibility to configure for internal routing in logic mode. A single tile can implement multiple logic functions on a single crossbar array. The output of one function can internally be routed to the input of another logic function.

LUT Based Approach
A closer examination of the internal design of the tile (as depicted in Figure 3.3) showcases a column of the crossbar array highlighted in blue. Here, each column symbolizes a separate logic function.
An LUT-based approach offers better scalability and speed than methods that convert combinatorial logic to sequential AND, OR or NAND operations. Compared with the sequential placement of boolean functions, LUTs reduce sequential steps, thus shortening the amount of routing needed. An operation like XOR in Liquid Silicon would be unfolded in three sequential steps, requiring two tiles to execute the required steps in sequence. However, with a LUT, results are ”pre-computed” and stored in memory and can be instantaneously retrieved by selecting the correct row in the crossbar array. According to Chen et al., combining internal routing with a LUT-based approach requires less hardware to implement a more complex electrical circuit.
Chen, et al., further improved the architectures with two techniques, only possible with eNVMs memory devices. Firstly, they utilize ReRAM devices as switches (Also discussed in Section 3.1.2), catering to both internal routing and intra-tile connections. Reducing the dependency on controlled CMOS transmission gates and offering a non-volatile routing solution. Secondly, they distinguish the directionality of information flow within the crossbar array between operational modes. In Logic and Memory modes, currents are initiated by a latching sensing amplifier, directing it through the ReRAM cell to the ground. However, in Interconnect mode, this latching is disabled, utilizing the pre-existing level restorer of the SA.

Results
The findings presented by Chen et al. are compelling. They report a power-delay product (PDP) reduction of 7-8X compared to SRAM-based FPGAs and 5-7X relative to Liquid Silicon. However, it is crucial to note that their results are preliminary, derived from simple benchmarks rather than the exhaustive tests that validated Liquid Silicon’s claims.

3.2.3. Discussion on Liquid Silicon and Merging Everything
In our analysis of eNVM-based Reconfigurable Computing Architectures (RCA), two architectures emerge as primary subjects of interest: Liquid Silicon and Merging Everything, as elaborated in Sections 3.2.1 and 3.2.2. These architectures operate on the principle that tiles can be reconfigured at the architectural and the implemented digital circuit level.
Central to both architectures is the ReRAM-based crossbar array. Liquid Silicon utilizes the inherent ability of the ReRAM crossbar array to perform logical NAND operations. In contrast, Merging Everything employs a LUT-based approach, storing pre-computed values in the grid.
Liquid Silicon offers a direct internal structure suitable for executing basic combinatorial logic functions. For more complex functions there is a requirement to use multiple tiles to implement the desired logical operations. This inefficiency of the Liquid Silicon approach is also evident from Figure 3.4: the architecture depends on rerouting signals to other tiles to implement the more complex functions. Another issue, highlighted by Figure 3.4a, shows an underutilization of the ReRAM crossbar array when the output of multiple logic functions go in different directions.
Additionally, Figure 3.4b presents the necessary steps for using a tile in memory mode. This process requires three additional tiles to generate the appropriate signals for extracting data from a crossbar array in a usable format. The research does not provide a clear framework for scaling individual memory blocks. The indication is that a single memory block’s size is influenced by the total size of the memory array, but details on this mechanism are not provided. When a 16x16 grid is used for each tile, the maximum amount of storage is 256 bits. The additional tiles needed to support the address decode and column select need to expand to a 512-bit memory block.
Another concern is the fixed size of the NxN crossbar array which can not be changed after productin. An explicit dependency exists between implementing logical functions and the word size in memory mode. Several columns will remain unused in logic mode when opting for a larger crossbar array size to accommodate more memory.
On the other hand, Merging Everything aims to address these shortcomings. Using the LUT method, commonly used in FPGAs, and adding support to cascade these logic functions shows potential advantages. However, this approach relies heavily on multiplexers. For a 16x16 grid, the area demands are significant. Considering the peripheral components alone, the number of ReRAM cells and transistors is expected to rise substantially, especially when more complex internal routing is considered.



(a) Logic & Interconnect **(b)** Memory Mode

Figure 3.4: Liquid Silicon operations from [22]

3.2.4. Concluding remarks

Analyzing eNVM-based RCAs, both Liquid Silicon and Merging Everything stand out as significant architectures. Liquid Silicon offers a straightforward but limited approach, due to this approach there are concerns about the scalability. On the other hand, Merging Everything, with its LUT-based structure, promises a more robust solution for implementing logic. Despite its advantage, the heavy reliance of Merging Everything on multiplexers, leading to an increased count of ReRAM cells and transistors, does bring up concerns for the size of a single tile. Weighing the merits and concerns, Merging Everything, with its superior performance and scalability, emerges as the more favourable choice among current state-of-the-art eNVM-based RCAs.

3.3. eNVM Components in FPGA Design

The prior sections examined architectural RCA designs integrating eNVM technologies. However, beyond these broad architectural shifts, there is also merit in examining advancements to components used explicitly in FPGAs. In FPGA design, the capabilities and efficiency of the overall system are heavily influenced by its foundational components. This section delves deep into the pivotal role played by eNVM technologies in improving these core FPGA components. Our discussion primarily centres on two fundamental FPGA components: LUTs and Multiplexers. Incorporating eNVM technologies into these components can substantially improve energy efficiency, presenting more optimized design solutions without sacrificing performance.

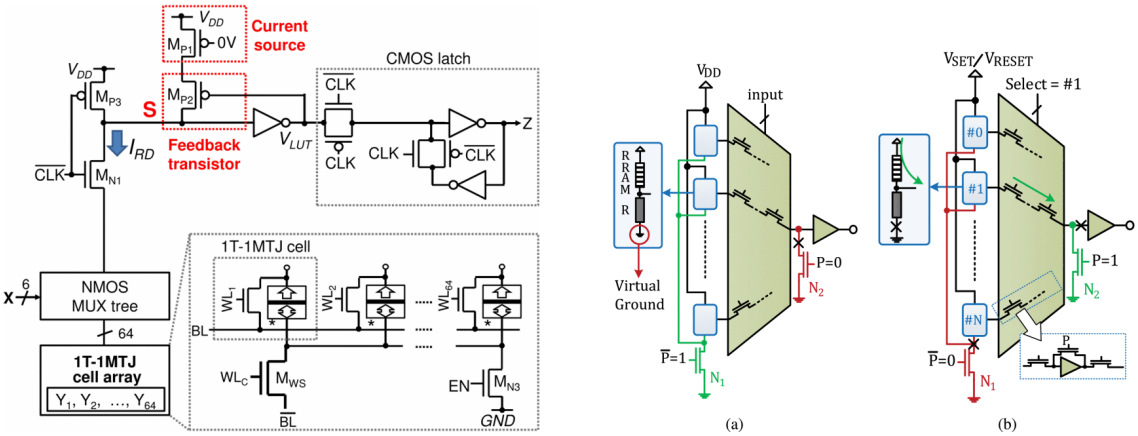
Look-Up Tables

Emerging eNVM technologies such as STT-MRAM and FeFET show promising results in improving power efficiency and reducing component count in FPGA designs. Suzuki et al. [12] utilized STT-MRAM (MTJ) technology in their design, which has influenced subsequent works like Chen [17]. Their approach, depicted in figure 3.5a, focused on increasing speed by implementing an improved single-ended SA design. As a result, the MTJ-based LUT design was able to reduce transistor count by 2.5X while delivering a 1.8X boost in PDP.

Building on the concepts of [12], Chen [17] proposed a design that uses FeFET technology to replace SRAM. Their approach maintained a similar transistor count while achieving an average power reduction of 1.7X. FeFET technology also showed a slight performance advantage over comparable ReRAM-based designs.

In contrast to the above designs, Khaleghi [24] proposed a ReRAM-based LUT design, as illustrated in figure 3.5b. Their approach did not employ a clocked SA, making their architecture more similar to traditional SRAM configurations.

Incorporating advanced eNVM technologies in LUT designs presents promising alternatives to traditional SRAM-based FPGA designs. The improvements in power efficiency and component count make them worthy of further exploration in future research. For our research we can learn from the used sensing circuits and the implications of these components on energy usage. FeFET in particular in the work of [17] has shown that there design is particular adept to be used as an alternative for SRAM-based LUTs.



(a) STT-MRAM LUT implementation from [12]

(b) ReRAM LUT implementation from [24]

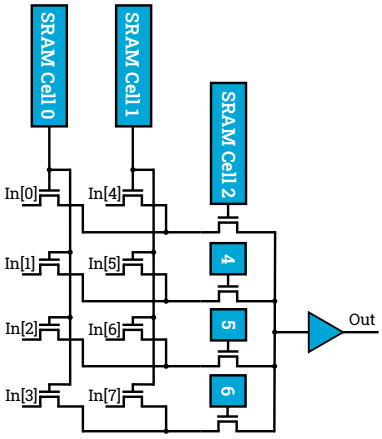
Figure 3.5: LUT implementations with emerging eNVM

Multiplexers

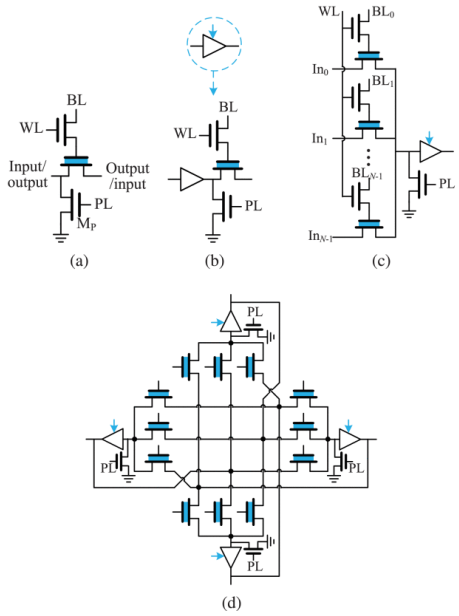
Multiplexers (MUX) in FPGAs execute tasks such as selecting specific memory cells, as indicated in figure 3.5a and 3.5b, and directing the data flow. In conventional SRAM-based FPGAs, MUXes handle signals in the local interconnect of a CLB. Figure 2.3a shows a typical implementation using a crossbar of pass transistors controlled by an SRAM cell.

Emerging eNVM technologies, like FeFET and ReRAM, present an opportunity to improve MUXes. Their inherent ability to maintain a state means they can be placed directly in the signal's path, elim-

inating the traditional mechanism where memory technologies control a pass transistor. This direct approach offers a more elegant solution with potentially reduced latencies and energy consumption.
The 'Merging Everything' design, presented in [23], demonstrates the capabilities of ReRAM in routing. Here, MUXes, fundamentally constructed from ReRAM cells, play a crucial role. These MUXes can be reconfigured by altering the state of the ReRAMs, which sets data paths, providing a flexible yet non-volatile routing mechanism.
Moreover, [17] introduces an alternative solution, offering FeFET-based routing elements as substitutes for standard CMOS-based routing. As illustrated in figure 3.6b, this FeFET approach reduces the overall transistor count, paving the way for a more compact FPGA design and enhanced efficiency.
In conclusion, the emergence and integration of eNVM technologies, like ReRAM and FeFET, into FPGA architectures can improve components used for routing signals through the system. Multiple studies, including the ones discussed, underscore the potential and versatility of eNVMs in replacing the traditionally used SRAM-controlled MUXes. These innovative eNVM-based solutions provide inherent non-volatility and suggest some advantages in power efficiency.



(a) SRAM based MUX used as CB



(b) FeFET based routing from [17]

3.3.1. Concluding remarks

In evaluating the role of eNVM technologies in RCA design, particularly within the contexts of LUTs and MUXes, the potential for optimization becomes evident. Integrating ReRAM and FeFET into these core components can reduce transistor counts and enhance power efficiency. Furthermore, the inherent non-volatility offered by these eNVM technologies provides distinct advantages over traditional SRAM-based solutions. The studies discussed in this section illustrate the evolving landscape of RCA design and highlight the benefits of incorporating emerging eNVM technologies.

3.4. Conclusion

Emerging eNVM technologies have showcased significant potential in reshaping the landscape of reconfigurable computing architectures. Emerging eNVMs are non-volatile, meaning they can retain data even when the power is switched off, providing faster set-up times upon system startup. Other benefits are that fewer components are needed and that they are more energy efficient.

Various eNVM technologies, such as STT-MTJ, ReRAM, and FeFET, have emerged as promising candidates to replace conventional memory technologies within computing systems, given their unique electrical characteristics. STT-MTJ is especially notable for its better endurance over the other technologies, making it well-suited for applications with frequent memory operations. ReRAM, with its two-terminal configuration, offers some unique circuit design implications that complement various routing tasks. On the other hand, while FeFET faces endurance-related challenges, performance metrics across other areas may indicate the best performance in speed and energy efficiency. All these technologies are inherently suitable for non-volatile memory storage. While each has strengths that distinguish it from the others, it is crucial to note that these technologies are still under active research and development, with the potential to be further improved.

Emerging eNVM technologies can also be used to innovate reconfigurable computer architectures. Some novel state-of-the-art architectures developed using these technologies include Liquid Silicon and Merging Everything.

The Liquid Silicon design utilizes a 2D homogenous grid of tiles built with at its core a memory array utilizing emerging eNVM. Implementing a homogenous grid, which can define routing, memory, or logic resources at a higher architectural level, is a significant improvement over some traditional designs. Because this approach enables the reconfiguration of resources based on the requirements of an application. However, it is the Merging Everything architecture that stands out prominently.

Merging Everything implements similar high-level concepts with reconfigurable resources at an architectural level. However, the implementation of logic functions is noticeably dissimilar. This architecture prioritizes LUTs at the core of the tile. Unlike direct implementations using logic gates that might require an increasing number of sequential steps for more complex functions, a LUT-based approach, as adopted by Merging Everything, remains consistent in their hardware demands. This consistency, combined with the extensive internal routing inside of a tile, shows an advantage over Liquid Silicon in size and power efficiency.

In summary, eNVM technologies, while still under active research, showcase their potential in reshaping reconfigurable computing architectures. Integrating these technologies can improve electrical performance, reduce device count, and offer non-volatility. Given the consistent progress in eNVM technologies and their growing influence on computational architecture designs, it is evident to research their use in RCA design.

4

Unified-Tile

This chapter presents a novel fine-grain RCA leveraging FeFET non-volatile memory technology to enhance size and power efficiencies compared to traditional FPGAs. Section 4.1 outlines the design objectives. Section 4.2 provides an architectural overview of component structuring and function implementation. Section 4.3 details the electrical implementation. In Section 4.4 we provide a conclusion of this chapter where discuss achievements of our architecture.

4.1. Design Objectives

This section outlines the objectives guiding the design of the novel fine-grain RCA. Fine-grain RCAs are designed to be versatile and adaptable, allowing the implementation of various digital circuits. To achieve this flexibility, fine-grain RCAs comprise several core components, each serving a critical function in the overall architecture. These core components are fundamental to the structure and operation of a fine-grain RCA and the proposed design in this thesis.

Core components of a fine-grain RCA: Given the extensive introduction to the design of FPGAs in the prior sections, it is essential to underline the core components that define a fine-grain RCA like an FPGA. Building upon the insights shared in Section 2.1, these integral elements are essential for implementing a fine-grain reconfigurable architecture like an FPGA and must be incorporated into our design.

- **Configurable Combinational Logic:** As demonstrated with FPGAs, this refers to adaptable logic gates and circuits, creating various digital functions per user requirements.
- **Configurable Interconnects:** These are the integral channels that facilitate signal routing between different logic blocks and between the logic and I/O interfaces, serving as the data highways within the architecture.
- **Memory Blocks:** Analogous to the EMBs in FPGAs, these units are designed for temporary or permanent data storage, optimising data access times and fulfilling data storage prerequisites.
- **Input/Output Interfaces:** Mirroring the IO interfaces of an FPGA, these entities manage bi-directional data and signal exchanges between the reconfigurable computing platform and its external environment.

While many contemporary FPGAs incorporate Digital Signal Processing (DSP) elements to elevate the operations of common signal processing tasks, including these advanced components will not be addressed in this thesis. Such complexities, though enhancing processing speeds of modern FPGAs, lie beyond our present scope of exploration.

4.1.1. Overarching design objectives:

The primary aim is to develop an architecture implementing the presented core elements while leveraging emerging non-volatile memory technology. Merely replacing the SRAM cells of traditional FPGA building blocks is not thorough enough. We also need to consider the potential of non-volatile memory

technologies to improve RCA design.
The specific objectives include:

- **Utilisation of emerging eNVM:** Utilise the properties of FeFET eNVM, such as non-volatility, low power consumption, and high speed, in constructing the fine-grain RCA core components.
- **Beyond SRAM Replacement:** Develop an architecture that does not merely substitute SRAM cells with FeFET eNVM but also leverages non-volatile memory advantages.
- **Performance Improvement:** Create an architecture meeting traditional FPGAs' functional requirements while improving processing speed and power usage.
- **Reconfigurable Resources:** Create a reconfigurable architecture which can reconfigure the allocation of used resources on the fabric.

The subsequent sections provide a detailed discussion of the specific objectives mentioned above.

Utilisation of emerging eNVM This thesis explores the potential of emerging non-volatile memory technologies, specifically FeFET, STT-MRAM, and ReRAM, for their application in fine-grain RCAs. The proposed design integrates these non-volatile memory technologies into the core components as alternatives to traditional SRAM cells. While the primary focus is structuring the design using FeFET technology, a comparison between STT-MRAM, ReRAM, and FeFET will also be included. Furthermore, subsequent sections will discuss the necessary modifications required to accommodate the distinct sensing requirements of these non-volatile memory technologies.

Beyond SRAM Replacement While several studies have explored replacing SRAM cells with newer memory technologies, this work aims to utilise these technologies in novel ways, particularly in the configuration memory of reconfigurable elements and internal routing components.

Reconfigurable Resources Traditional FPGAs are constrained by their island-based topology, resulting in increased latency and power consumption due to the necessity of data movement across varying distances on the fabric caused by the fixed physical locations of memory elements. One of the objectives is to develop an adaptive resource topology optimising data movement and resource allocation. This involves leveraging non-volatile memory properties to implement a more adaptive interconnect structure characterised by a homogeneous mesh of elements (re)configurable to implement the fine-grain RCA's core components, thereby enabling topology reconfiguration based on specific applications.

Summary
This section outlined the primary objectives for our design, emphasising the importance of incorporating FeFET eNVM into the core components. The architecture was devised by not merely replacing the SRAM cells with FeFET eNVM but also harnessing the advantages of non-volatile memory. This approach aimed at boosting performance and formulating a reconfigurable architecture that optimised data movement and resource allocation.

4.2. System Architecture

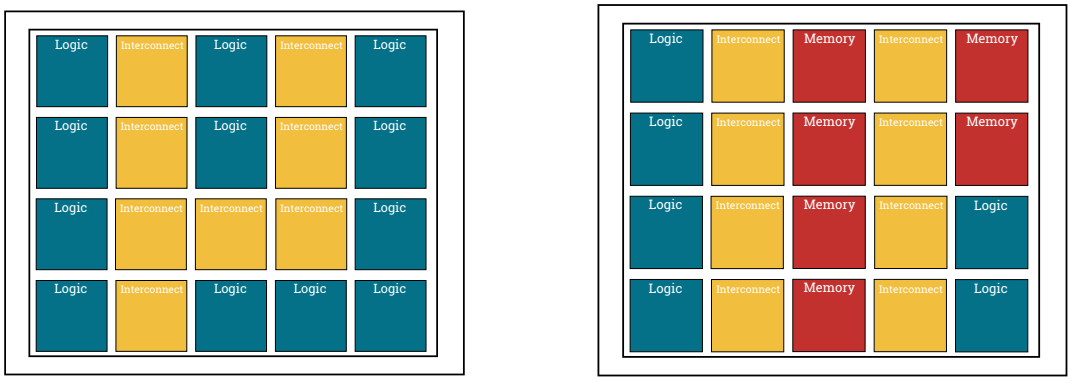
The system architecture of a reconfigurable computing system defines the organisation and interconnection of its components. The architecture has a significant impact on the performance, efficiency, and flexibility of the system. In this chapter, we explore the system architecture of a novel reconfigurable computing platform, the unified tile architecture. This architecture is in line with other recent developments of Liquid Silicon and Merging Everything [23, 20] reconfigurable computing, offering several advantages over traditional RCA like FPGAs. The following section will provide a detailed overview of the unified tile architecture, including its key features and benefits. We will also discuss some challenges with designing and implementing the unified tile architecture.

4.2.1. High-level Architecture

The architecture presented in this thesis, at its core, the design offers a unified tile-based approach, where a 2D grid of homogenous components is used. This homogeneous component called a tile integrates the core functionalities of an RCA.

Unified Tile

The core of the proposed architecture is the ”unified tile”. Unlike traditional designs, where the allocation of resources is rigid and defined during silicon production, this grid of reconfigurable homogeneous tiles offers the ability to tailor the amount of resources to the need of an application. Each tile can be reconfigured to implement logic functions, store memory, or route signals (interconnect). For illustrative purposes let us examine Figure 4.1, two images are given where resources are mapped. Figure 4.1a showcases a configuration where a unified tile is oriented towards combinatorial logic operations. In contrast, Figure 4.1b presents a configuration where the tile is optimized for memory-intensive applications. The inherent advantage of this architecture is its ability to reconfigure and adapt resource distribution. For instance, if an application necessitates more memory resources, the system can be reconfigured to cater to this demand.



(a) Unified tile in a logic-centric configuration

(b) Unified tile in a memory-centric configuration

Figure 4.1: Micro-architectural adaptability of the unified tile

The architecture uses a 2D grid layout, where each homogeneous tile, regardless of mode, supports some basic interconnectivity with neighbouring tiles. Each tile can interface with its adjacent tile in all cardinal directions - North, East, South, and West. The IO periphery can also directly connect to IO when a tile is situated on the borders of the architecture. Tiles can be reconfigured into an ”interconnections mode” to grant additional options for configuring datapaths in the system.

4.2.2. Architecture Unified-Tile

Figure 4.2 depicts the different components of the architecture of the unified-tile. From an higher architectural perspective, there needs to be routing to neighboring tiles. The neighbouring tiles are connect via configurable connections nodes, this allows the routing on bit-level from one of the four sides of a tile. Output from tiles and input stimuli control different components inside of the tile. The most important component is the crossbar array with the associated controlling components. The crossbar array is used to store the configuration data for the three different modes. The differences in modes arises from this data and the internal routing of signals.

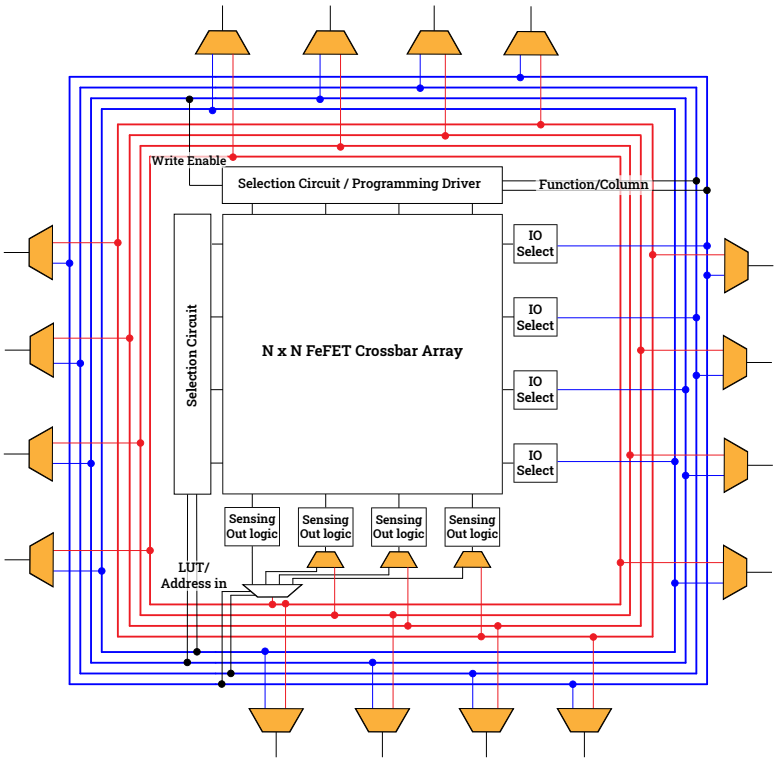


Figure 4.2: Architectural composition of the unified tile

Tile Outer Periphery The adjacent tiles connect with the periphery, controlling the path of signals are routing MUXes (indicated in yellow). The flow is controlled by selecting individual bits from one of the four directions. These bits serve as the control signals that can control different components of the tile. Each line in the figure symbolizes a bit, with blue lines representing input and red lines indicating output. In Section 4.2.6, these aspects of inter-tile communication are discussed in more detail.

FeFET Multiplexers eNVM can be utilized in more ways than solely in a crossbar array. One such application involves placing them directly in the signal paths, such as using FeFET MUXes instead of pass-transistor-based MUXes. This eliminates the need for actively controlling the signal flow from IO or memory. FeFET MUXes provide a non-volatile solution for selecting the signal path, which is a convenient option for routing. In the outer periphery, they enable the routing of bits from neighbouring tiles to one of three choices: input, output, and high-Z state.

FeFET Crossbar Array Central to the tile-based architecture is the FeFET Crossbar Array, serving as the central component of the tile. In the crossbar array the configuration data for all operational modes is stored. While the tile's structural framework remains constant, how information is mapped on the crossbar array, coupled with the flow of control signals, determines a tile's behaviour. Facilitating the three distinct modes: logic, memory, and interconnect. Two selection circuits are embedded on

the edge of the arrays, dedicated to row and column selection. Notably, one of these circuits boasts a programming driver which can change the state of the FeFET.
For **logic** operations, the crossbar array's ability to store the outputs needed, like in a LUT. This facilitates the creation of various digital functions and ensures logic implementation. Simultaneously, the very architecture of the crossbar array naturally lends itself to **memory** operations, given its intrinsic data storage capability. Finally, the crossbar array's structural layout supports signal routing. Enabling the configurations of pathways for **interconnection** of signals.

Auxiliary Logic and Output Mechanisms Adjacent to the rows of the tile is input logic, designed to enable external signals to interface directly with the rows. Section 4.2.5 provides a detailed discussion of this interaction. Each column incorporates a sense amplifier and auxiliary output logic. Currently, the only implemented auxiliary output logic is a flip-flop.

Internal routing The input ring lines, coloured blue, are pivotal in establishing connections among the tile's various components. For instance, the connection path to a selection circuit can not be changed, and some connections are associated with the selection circuit on the right. In contrast, other connections are associated with the top selection circuit.
In the subsequent sections, we will provide a detailed discussion of the different modes of the unified tile.

4.2.3. Logic Mode

In the Logic Mode, the tile is configured to implement boolean logic functions based on an N-bit input, leveraging the intrinsic $N \times M$ FeFET crossbar array. Note that the tile's architectural components remain consistent irrespective of its operational mode. The routing of the control- and data path, combined with a specific arrangement of data in the crossbar array, determines the tile's function.

Operation

The representation of logic behaviour using the memory crossbar array is depicted in Figure 4.3. In this mode, the tile's structure is reminiscent of a LUT. These results in this architecture are mapped onto the $N \times N$ crossbar array.
The crossbar consists of rows and columns. Each column comes with a sensing circuit, which determines the state of its corresponding memory cell. A feature of this architecture is that different logic functions can be mapped onto each column. For the following examples we use the following 4 different functions given in table 4.1.

Table 4.1: Mapped logic functions in Figure 4.3

Logic Function A	B	A XOR B Function 3	A NAND B Function 2	A AND B Function 1	A OR B Function 0
0	0	0	1	0	0
0	1	1	1	0	1
1	0	1	1	0	1
1	1	0	0	1	1

Figure 4.3a projects the table data onto the crossbar array, where the red-lettered inputs 'A' and 'B' in the figure mirror the values given in the table. The selection circuits engage based on these inputs to select the appropriate row (result) and column (function) in the crossbar array.

Execution Example

Figure 4.3b gives a practical rendition of this mechanism. For specific inputs, say $A = 1$ and $B = 0$, the OR operation is performed. Consequently, Function 0 alongside the third row is activated, producing the expected outcome $1 \vee 0 = 1$. utput values from the array are directed to lines equivalent to column numbers. The output of the SA is connected to the output, as depicted in Figure 4.2.

Additional components

Incorporating Flip-Flops after the sense amplifier allows the tile to retain diverse function outputs over consecutive clock cycles. As a result, previously obtained outputs remain intact, even when new operations are executed. Furthermore, by integrating an extra multiplexer at the outputs, controlled by the same signals as the top selection circuit, the system can effectively route the signal to a known bit position instead of the other configuration where every SA is connected to an individual output position. This implementation is represented in Figure 4.4. The benefit of adding this extra multiplier is that it simplifies the routing to other tiles, when a function is not mapped to a single column but to the whole crossbar array.

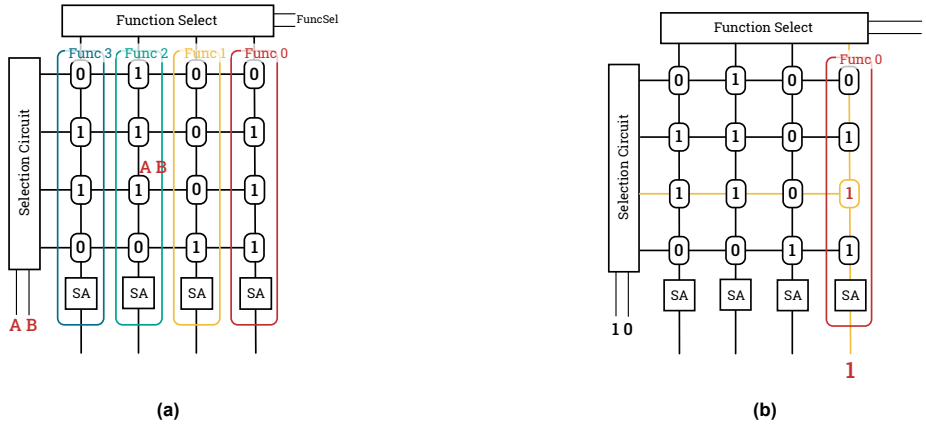


Figure 4.3: Simplified representation of components pertinent to logic mode

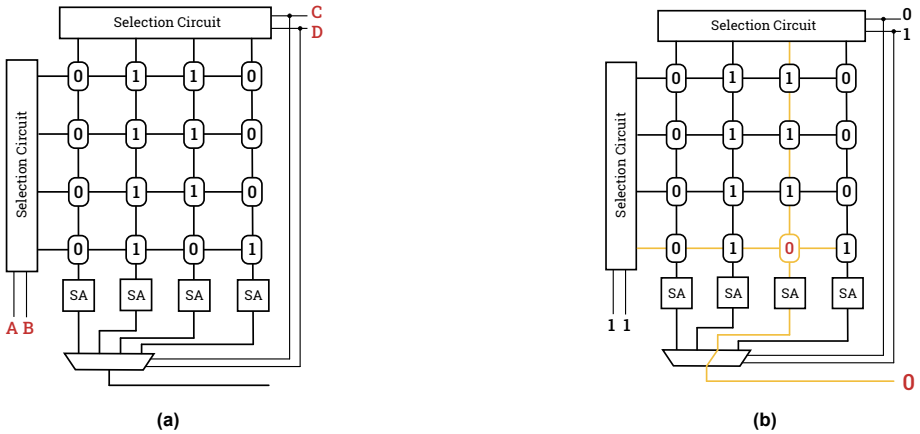


Figure 4.4: Simplified representation of logic mode in Full LUT configuration

Advantages of a crossbar array

Dividing the selection circuits into rows and columns (implementing a crossbar array) and positioning a multiplexer at the end exhibits clear advantages over storing data in a 1D array. This is evident from the increasing MOSFET count in a typical mux tree relative to the number of inputs. The necessary quantity of 2:1 multiplexers for an NMOS mux tree can be calculated using Equation 4.1. By segmenting the selection and adding a MUX to the end as represented in Equation 4.2, the total number of MUXes thus hardware is reduced. This visually supported by Figure 4.5 which plots these two functions.

$$N_{2:1MUX} = \left(2^{N_{in}} - 1\right) \tag{4.1}$$

$$N_{2:1MUX} = 3\left(2^{N_t} - 1\right),\ N_t = N_{in}/2 \tag{4.2}$$

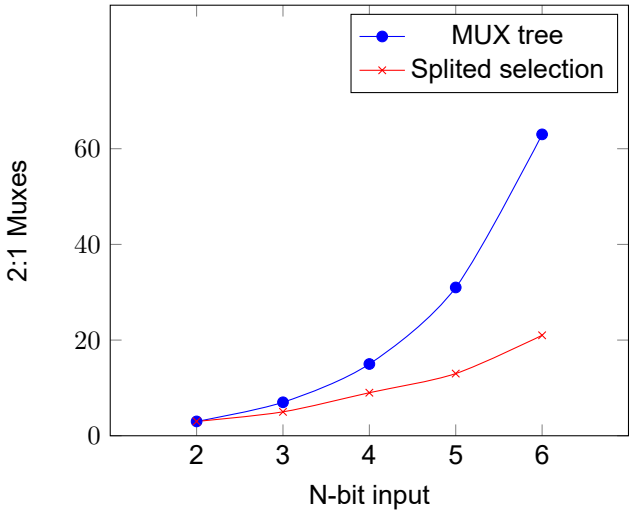
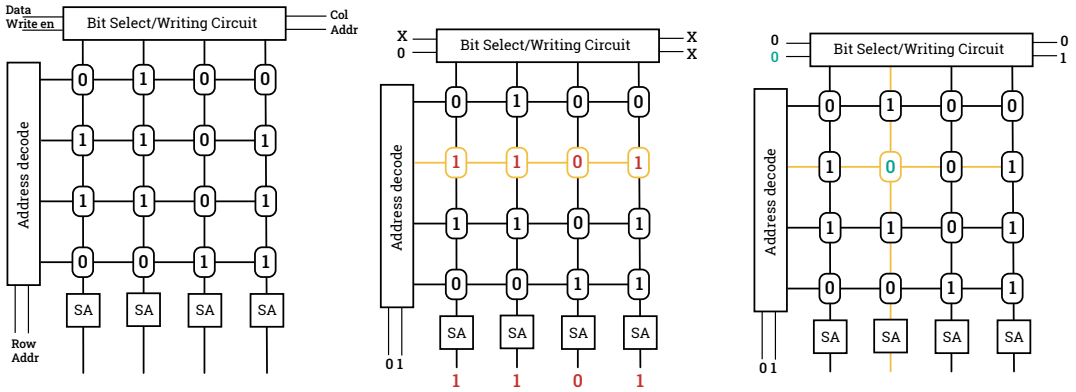


Figure 4.5: Line graph of the amount of 2:1 muxes needed for selecting a memory cell

4.2.4. Memory Mode

In memory mode the tile operates as an memory block.There are two core operations supported in memory mode: **Reading:** In Figure 4.6b a reading operation is depicted. Reading is similar to the operation that occurs in logic mode. The difference here is that where logic mode selects a single value. In memory mode the content of the whole line is read in a single instruction. The width of the output in reading mode is equal to the number of columns of the crossbar array. **Writing:** Figure 4.6c illustrates the writing process in the tile. During a writing operation the state of FeFET is changes meaning that first a memory cell is isolated and then a programming voltage is given to gate of the FeFET. Note: Given the tile's architecture, it is only equipped for single-port memory operations, it does not support concurrent read-write operations.



(a) Components essential for memory mode

(b) Line reading process

(c) Data writing process

Figure 4.6: Components and operations in memory mode

Interplay of LUT size and Memory size

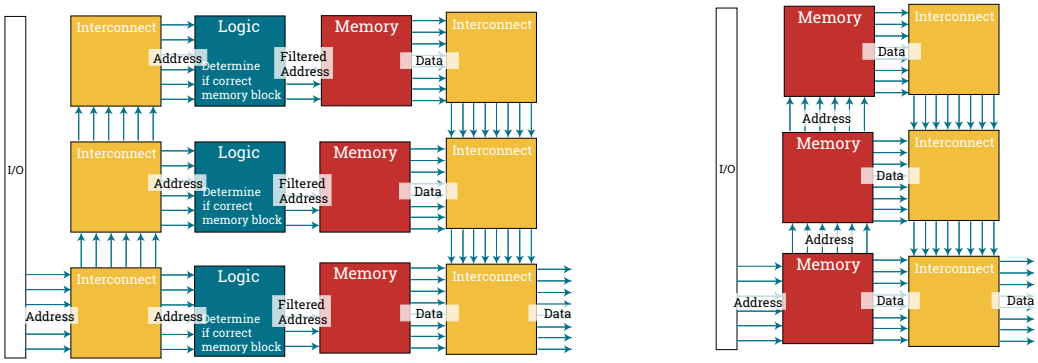
The crossbar array's design inherently provides memory storage. However, the optimal size for the memory array in logic mode may differ from that required for efficient memory storage. If the array is sized as a small input LUT, it might not be the most efficient for larger memory blocks, and vice versa. This trade-off of the multi-functionality of a tile is an inherent challenge in an architecture combining memory and logic operations.

Once set during fabrication, the tile's physical dimensions remain fixed, meaning the crossbar size cannot resized post-fabrication. A solution which alleviates a part of this problem is the interconnection

of multiple tiles operating in memory mode to create a bigger memory block. Figure 4.7a gives an example of such an expansion.

Interconnecting Memory tiles

With some additional hardware to the tile the connecting of multiple tiles together can be improved. Figure 4.7 illustrates the amount of tiles needed to interconnect multiple memory blocks which use the same address signal. In the first illustration there is no support for this inside of the tile. Which requires the use of logic configured tiles to filter the address to select the appropriate tile. In the second image we reduce the amount of tiles needed by introducing an AND gate combined with a configurable mask, as showcased in Figure 4.8. This allows the generation of an enable signal that facilitates synchronisation between tiles based on a shared address space.



(a) memory stacking without specialised hardware

(b) Efficient memory stacking

Figure 4.7: Comparing memory mode configurations

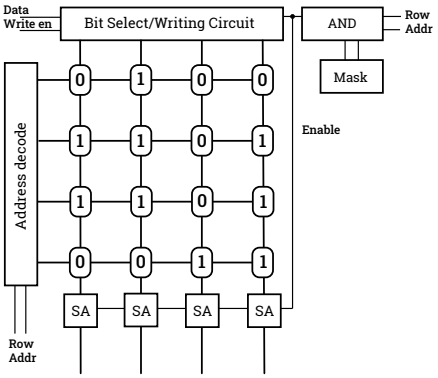


Figure 4.8: Hardware improvement for interconnected memory

4.2.5. Interconnect Mode

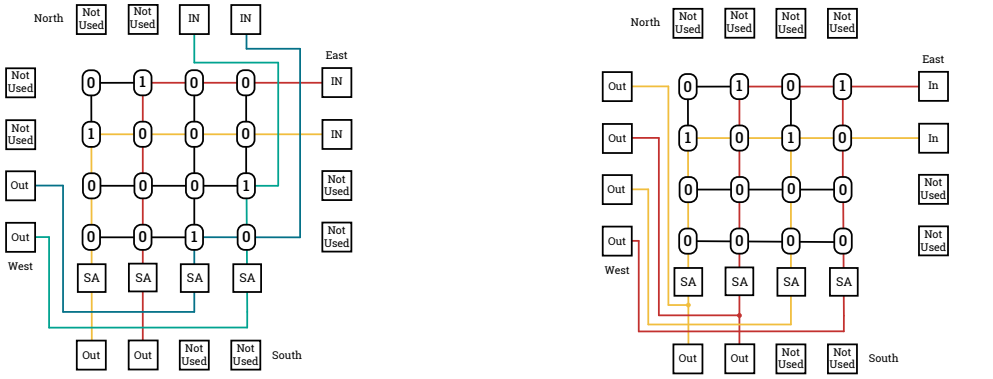
Within the tile-based architecture, every tile inherently provides basic routing functionality, allowing for direction-based signal routing. This built-in capability determines the flow across the cardinal directions: North, South, East, and West. However, for more intricate routing demands, especially when there is a need to manipulate both direction and bit position, tiles are needed, which are set in interconnect mode.

Functionality

Unlike logic and memory modes, the interconnect mode is tailored to give more options in signal routing. Figure 4.9 visually illustrates a tile in this mode.
The FeFETs in each row are connected to specific bit positions, with row 1 linked to bit 1, row 2 to bit 2, and so forth. Similarly, columns are linked with their respective bit positions, e.g., columns 1 to bit 1 and 2 to bit 2.
For the interconnect mode, the state of each FeFET, in conjunction with its associated row, forms the basis for signal routing. A FeFET in the 'On' state signifies an active route for the corresponding bit position. The SA associated with each column detects this 'On' state in a FeFET. This mechanism effectively evaluates the AND operation between the FeFET's state, its corresponding row, and the signal applied to that row, determining the routing path.

Examples

Let us examine the scenario depicted in Figure 4.9a: Signals enter the tile—two from the north at bit positions 3 and 4, and two from the east at bit positions 1 and 2. As an illustrative example, the signal's path at bit 1 from the east is traced. Upon interaction with the 4x4 crossbar array, this signal's bit position is changed to bit 3. Consequently, it exits from the south on bit 3.
Further complexity is depicted in Figure 4.9b. Here, signals from the east on bits 1 and 2 are rerouted to both the South and West on bits 3 and 4. Simultaneously, replicas of these signals are directed west, retaining their original bit positions of 1 and 2.



(a) Example illustrating signal rerouting

(b) Demonstration of signal duplication and rerouting

Figure 4.9: Scenarios underscoring the capabilities of interconnect mode

4.2.6. Inter-tile Communication

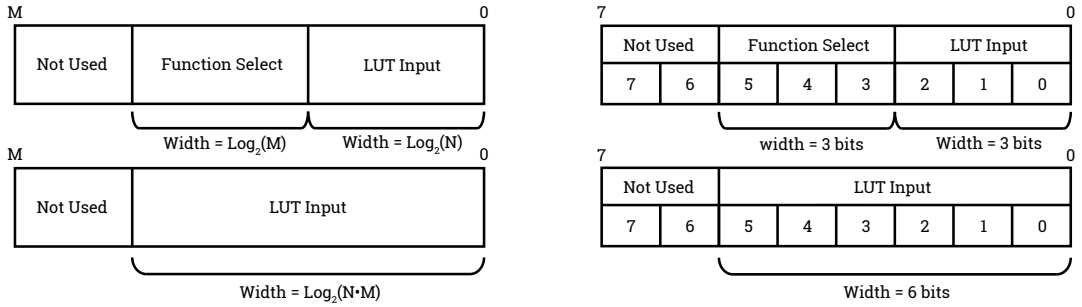
While we primarily focused on individual tile behavior in our architectural exploration, inter-tile communication does have its constraints. A set of defined instructions governs inter-tile communication. The bit positions dictate how signals control various components of a tile. Let us examine this aspect for the different operational modes.

Logic Mode

Every tile is designed with specific signal pathways. As depicted in Figure 4.3a, in logic mode, the signals `FuncSel` and `A B` control specific components. `FuncSel` controls the upper selection circuit, deciding which column gets activated for logic operations. On the other hand, `A B` acts as the input for the left-side selection mechanism, serving the LUT. For a grid characterised as NxM we characterize N as the width of the grid and M as height. Figure 4.10a showcases the general layout for which the bit positions control specific functions in the set mode.

- **LUT Input:** Spanning a width of $Log_2 M$. starting from the least significant bit and continues to its defined width.
- **Function Select:** Its width is $Log_2 N$. positioned after the LUT input's most significant bit, it spans its designated width.

For a practical perspective, the instructions for an 8x8 crossbar array are depicted in Figure 4.10b.

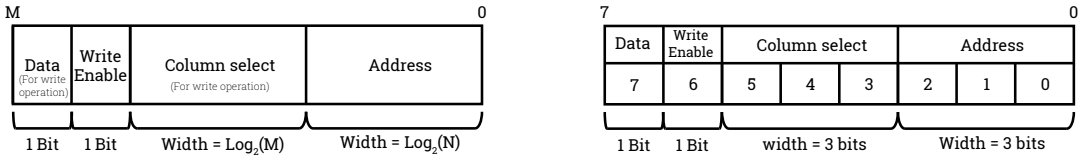


(a) General solution **(b)** Bit positions 8x8 array

Figure 4.10: Bit positions for logic mode

Memory Mode

Memory mode operates with four key signals: `Data`, `Write`, `Enable`, `Row Address`, and `Column`. Figure 4.11a shows the general bit positioning for the instructions for memory mode. Note that the bit positions for column and row selection are identical in memory and logic modes. This similarity arises because these signal paths control the same internal components in both logic and memory modes.
The Write Enable and Data signals follow the most significant bit of the column select. A detailed bit positioning for an 8x8 grid is provided in Figure 4.11b.



(a) General solution **(b)** Bit positions 8x8 array

Figure 4.11: Bit positions for memory mode

Interconnect Mode

In the interconnect mode, there is no requirement for a bit to control a specific component. However, the direct association of bit position 0 with row 0 of the crossbar array is essential. If a signal needs to traverse this row, its corresponding bit position should be 0.

4.2.7. Conclusion

We have introduced a novel RCA architecture that employs a 2D layout of homogeneous tiles—drawing inspiration from the Liquid Silicon and Merging Everything architectures. This approach utilizes a 2D layout of homogeneous tiles, with a unified tile at the center of the design. The unified tile combines three key functionalities of an RCA into one reconfigurable component.
With the ability to operate as a configurable logic block, a memory block, or a switch block, the unified tile can be connected to other tiles or IO. Each tile is designed to support routing to these connections, with designers able to control the bit level for routing specific signals to components within the tile.
The logic mode is implemented by mapping pre-calculated results on the crossbar array. This means that a given binary input is decoded to a position on the crossbar array, which contains the output for the mapped function on the tile. In memory mode, a row is selected from the crossbar array, with the option to change values inside the array, supporting read and write operations.
In interconnect mode, the crossbar array grid can be used to determine the path of signals. Combining these three modes allows for the implementation of any digital circuit, making this architecture a fine-grain RCA.

4.3. Implementation Unified-Tile

This section will discuss some electrical implementation of components of the proposed Unified-Tile architecture, specifically highlighting its memory structure. We will also describe the crossbar array and how we can operate the most important components associated with that array.
While the concepts presented here represent a viable solution, numerous alternative implementations exist for each component. The choice to focus on these specific designs stems from their balance of efficiency, innovation, and simplicity. However, an in-depth analysis of other circuits is beyond the scope of this discussion.

4.3.1. Crossbar Array

The crossbar array stands central to each tile's operation. This array structure can be considered a matrix, where multiple memory elements are ordered in rows and columns and can be individually addressed via word lines and bit lines. These word and bit lines are distributed horizontally and vertically. The crossbar's design offers several inherent benefits. Its tight packing of memory elements ensures a high storage capacity within a minimal footprint, leading to optimised space utilisation. Additionally, the shared word and bit lines streamline the addressing mechanism, reducing the requisite hardware and potentially conserving power. The straightforward pathways between memory elements and control lines eliminate the need for intricate routing, ensuring rapid and efficient memory access.
Three central components are associated with all operations performed on the crossbar array:

1. **Memory Element**: A combination of an access transistor and a storage element in implementing a FeFET device.
2. **Selection Circuits**: These circuits address and control access. Interacting with the word and bit lines activates a specific memory element during read or write operations.
3. **Sense Amplifier**: The sense amplifier determines the state of the memory element.

In the following sections, each of these components will be elaborated upon, providing an understanding of their respective function and operation of the crossbar array.

Memory Element

We explored several relevant memory technologies in Section 2.2. One of the overarching objectives of this study is to investigate architectures based on emerging eNVMs. From this directive, our presentation is based on the FeFET technology.

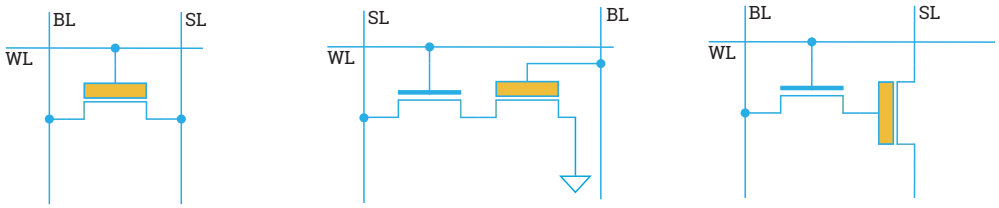
FeFET configuration in Crossbar Arrays: The FeFET technology has been explored for its potential in crossbar arrays, as evidenced by the study in [25]. The configurations from this study have been illustrated in Figure 4.12, among which three are particularly notable:

- *1T Design*: Figure 4.12a presents this design as the most area efficient. Featuring only a FeFET device, it adopts a dual-line control through WL and BL. The sensing of the state goes through the

SL. While efficient in space, this design has been indicated to face potential disturbances during read-and-write operations.

- *2T Designs*: This configuration can be divided into two subtypes, each with separate reading and writing paths. Variant (b) presented in 4.12b is preferable among them. It ensures that all control signals are directly interfaced with the transistor gates. The 2T/C design, according to [25], offers a more balanced approach regarding voltage range, write disturbances, and overall memory access energy and latency.

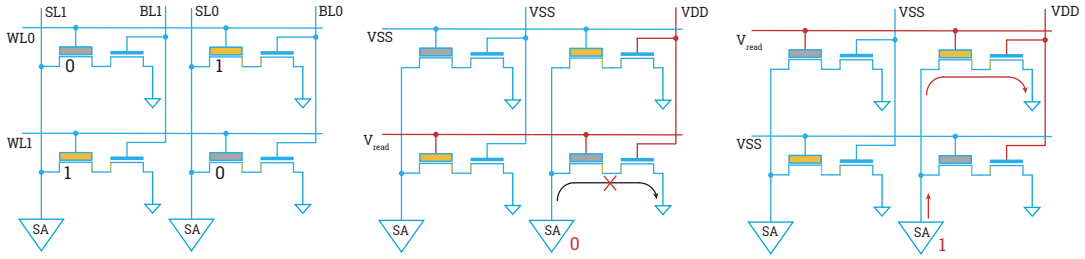
While there is mention of a 3T/C design in [25], we decided not to implement it in our research. The 3T/C configuration, though potentially offering a balanced voltage range and reduced disturbances, increases the transistor count without providing a clear advantage over the 2T/C design.



(a) 1T **(b)** 2T **(c)** 2T

Figure 4.12: Possible arrangement FeFET memory element

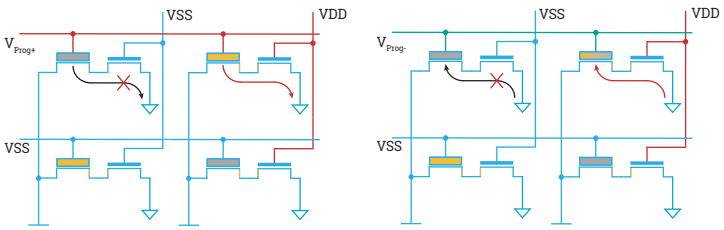
Understanding Read/Write Operations:
Read operation: Illustrated in Figure 4.13, sensing the state of a specific cell necessitates both row and column activations. The reading operations involve applying VDD to the NMOS gate and applying V_{read} to the gate of the FeFET. Subfigures 4.13b and 4.13c show the differences in dynamics when the FeFET is either set in a '0' or '1' state, highlighting the resulting current flow.



(a) Example Circuit **(b)** Read '0' **(c)** Read '1'

Figure 4.13: Example crossbar circuit (grey is in 0 state, yellow is in 1 state)

Write operation: As outlined in Figure 4.14, the writing operation mandates a precise voltage differential between the gate and drain to cause a state transition in the Ferroelectric layer. To perform this transition, the signal line must be set to a floating state (0), and the FeFET gate voltage must be set to a programming voltage associated with the desired state. Subfigures 4.14a and 4.14b respectively depict the transition protocols for the '1' and '0' states.



(a) Write '1' **(b)** Write '0'

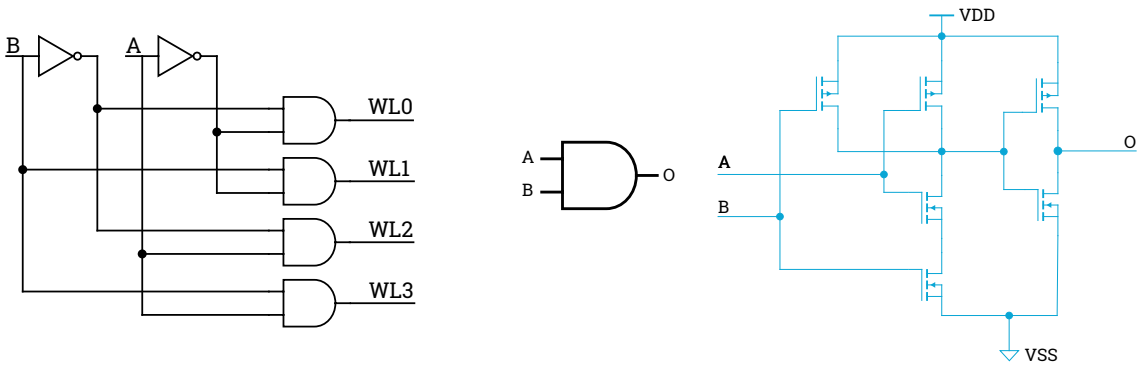
Figure 4.14: Writing operation in a crossbar array (grey is in 0 state, yellow is in 1 state)

4.3.2. Selection Component

This section presents a detailed exploration of the primary selection circuits integral to the tile architecture: the row selector (address decoder) and the column selector (programming/address decoder). These modules are based on CMOS technology.

Row Selection

As illustrated in Figure 4.15a, the row selection circuit utilises a logical combination of AND and NOT gates. This configuration is extendable to decode any N-bit input to select an individual line. The implementation of the CMOS gate is depicted in Figure 4.15b.



(a) 2-bit input selection circuit. **(b)** CMOS-based 2-bit input AND gate.

Figure 4.15: Schematic of the selection circuitry.

In operational terms, based on the n-bit input, the circuit can activate a single line. For instance, line WL_1 is activated with a high signal (logic 1) on input 'A' and a low signal (logic 0) on input 'B', adhering to the logic $WL_1 = A \wedge \overline{B}$.

Column Selection

The column selection component introduces additional requirements compared to the row selection because of the need to control voltage levels to perform reading and programming operations on the FeFET device.

Figure 4.16 depicts the refined column selection circuitry. The critical challenge here is to regulate the bit line voltages. This is achieved using a variable voltage source configured to generate the required voltages. While the detailed design of this voltage source is beyond the scope of this thesis, notable enhancements to the selection circuit include the integration of a transmission gate controlled through a mechanism similar to the row selection module.

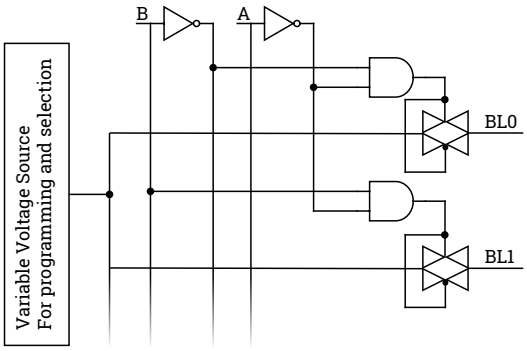


Figure 4.16: Column selection and programming circuit diagram.

4.3.3. Sense Amplifier

The Sense Amplifier converts analog signals into digital signals. In SRAM technology, a differential SA leverages the small voltage difference between paired bit lines to determine the memory cell's state. While differential SAs have advantages in SRAM technology, eNVMs like FeFET present different challenges.

Differential SA: Differential SAs amplify small differences in current between two lines. Direct interfacing with SRAM cells is straightforward due to its intrinsically differential pair of connection nodes of the SRAM cell. However, interfacing with eNVM technologies, like FeFET, which are predominantly single-ended, requires modifications and the inclusion of a reference current, as depicted in Figure 4.17, based on the design in [26].

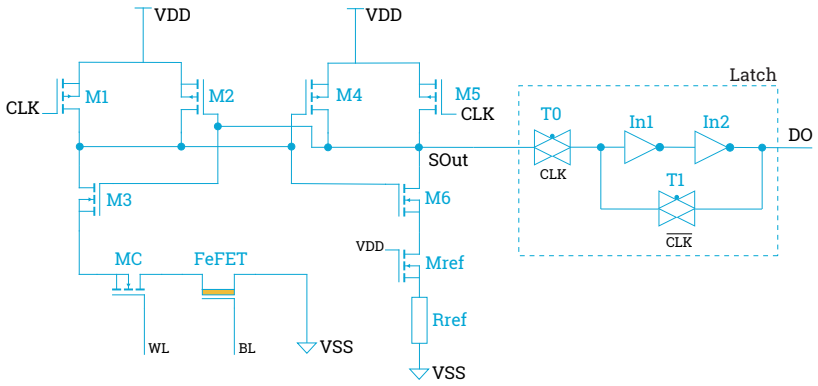
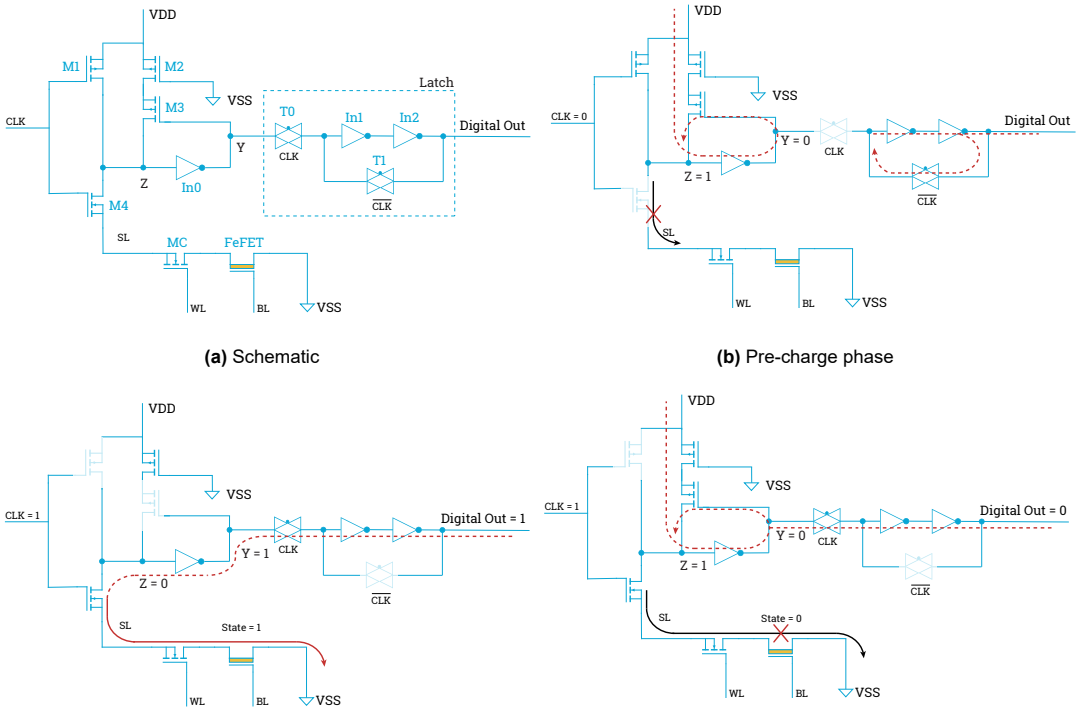


Figure 4.17: Differential Amplifier

Single-ended Latching SA: In contrast, single-ended SA, as proposed in [17], presents several advantages when integrated with FeFETs. Notably:

1. **No Reference Current Requirement**: It eliminates the necessity for a reference current to ascertain the FeFET state, simplifying the design.
2. **Reduced Overhead**: Owing to its 2T architectural configuration, the single-ended SA comprises fewer transistors, reducing the overall component count and, in turn, the power consumption.
3. **Adaptability**: The single-ended SA can efficiently address the sneak path issue, where a static current could flow even when it is not desired. Although this is a challenge, [17] suggests that the static power this introduces might still be lower than the dynamic power introduced by differential SAs' reference path.
4. **Potential for Further Optimization**: As mentioned in [17], there are possibilities for static power optimization in FeFET-based designs that can further enhance the efficiency of the single-ended SA design.

The single-ended SA's detailed mechanism is illustrated in Figures 4.18a to 4.18d, showcasing its operation across various phases and FeFET states. While it's true that this type of SA faces challenges when the ON/OFF current ratio is low, optimizations, like adjusting the size of certain transistors, can help mitigate these limitations.
While differential SAs do not face the static current issue in single-ended designs, they come with drawbacks when integrated with eNVMs. They require a reference current, which can increase dynamic power consumption. On the other hand, single-ended SAs, despite the sneak path issue, may not necessarily consume more power than their differential counterparts, as suggested by [17]. Given these mentioned factors, the single-ended latching SA is preferred for its simplicity, adaptability, and potential for further optimization in the FeFET environment.



(a) Schematic

(b) Pre-charge phase

(c) Evaluation phase with a '1' read

(d) Evaluation phase with a '0' read

Figure 4.18: Detailed schematic of the sense amplifier, as adapted from [17].

Operation of the Single-ended SA The design depicted in figure 4.18a has three distinct elements: a clock-driven pull-up circuit, a level restorer and a latch. M1 and M4 function as the pull-up circuit where M1 is active when the clock is low, and M4 is active when the clock is high. M3 and In0 form the level restorer, and as explained by Chen in [17], this circuit prevents the node entering In0 from floating when the path to VSS is blocked in case of a FeFET in the '0' state. The final element is formed by T0, In1, In2 and T1 elements form the latch, which can store the measured data between evaluation phases.
Figure 4.18b shows the current flow when the circuit pre-charges the sense lines. The level restores the pull-up of the circuit at this moment, which will generate a digital '1' at node Z, which is inverted and presented to transmission gate T0 at Y. When the clock is high, the SA goes into the evaluation phase. When the FeFET is in the '1' state, as depicted in figure 4.18d, the resistance to VSS is low; thus, the voltage at node Z is pulled down to zero. Z gets inverted into a digital '1' and will be stored in the latch. In case the FeFET is in the '0' state, the resistance to the ground is too high and node Z is not pulled down, resulting in a digital '0'

4.3.4. Routing MUXes

In Section 3.3 we provided a detailed exposition of the role of MUXes in directing data flow. Emerging eNVMs can be used to reduce the need for conventional techniques where SRAM cells control crossbar of pass transistors. The ReRAM-based solution presented in [23] and the FeFET-based solutions from [17] reduce area and energy consumption reductions.
In figure 4.19, an illustration is given for a solution. This includes the expected transistors required for programming. This implementation is used for the yellow indicated MUXes in Figure 4.2.

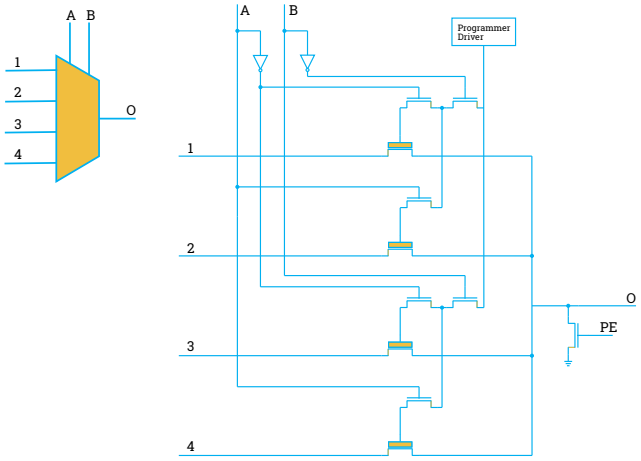


Figure 4.19: Routing MUX

4.3.5. Concluding remarks on implementation

The Unified-Tile architecture, as outlined in this Section, is a 2D homogenous tile architecture where each tile can be configured to implement the three main components of an FPGA. The crossbar array is essential for storing the data to implement the three modes. Various parts of the tile are implemented with emerging eNVM technology. In this Section, we used the FeFET technology to implement the circuits using eNVMs presented in Section 4.3. The goal of the presented circuits in this section is not to provide the most optimized results but to give some sample implementations which can be used to implement a tile architecture like this.
As the main memory element, the 2T design for eNVM cells is selected due to its technical advantages in controlling memory operations. As detailed in [25], the 2T/C configuration presents an optimal balance in terms of voltage range, minimized disturbance during write operations, and efficient memory access in both energy and latency aspects. This design can target specific memory cells within the crossbar array.
The crossbar array itself forms the heart of a tile, the peripheral circuitry needed to implement the required function is crucial. These circuits in the periphery consist of the selection circuits and sensing circuits. In this architecture, these components are constructed with CMOS.
Two prominent designs emerged in our exploration of Sense Amplifiers (SA): the Differential SA and the Single-ended Latching SA. While the former is a natural fit for SRAM due to its intrinsic differential nature, it poses challenges when interfacing with eNVM technologies like FeFET. Expressly, it necessitates extra circuitry and a reference current, leading to potential increases in dynamic power consumption.
On the other hand, the Single-ended Latching SA, as highlighted in [17], offers several advantages for FeFET integration. Beyond bypassing the need for a reference current, it showcases a reduced overhead with its 2T configuration. Even though this SA design grapples with limitations when the ON/OFF current ratio is small, it is the preferred choice for our FeFET-based architecture.
In conclusion, this chapter delves into the circuits used to implement components of the Unified-Tile architecture, showcasing a combination of CMOS technologies with emerging eNVMs like FeFET. While the designs presented are representative of the current state-of-the-art, it is important to note that better and more efficient designs are expected to be produced in the future.

4.4. Conclusion

This chapter introduced a novel RCA based on the FeFET eNVM technology. Beginning with an analysis of the FPGA, the most common RCA platform, we identified the primary functionalities essential to an RCA's platform. This understanding was therefore used in the development of our design, emphasizing the integration of these core functionalities while meeting the overarching design objectives:

- **Utilisation of emerging eNVM:** Capitalizing on the properties of FeFET eNVMs to construct RCA components.
- **Beyond SRAM Replacement:** Proposing an architecture that goes beyond simple SRAM substitution with FeFET eNVM, leveraging inherent technological advantages.
- **Performance Improvement:** Designing an architecture that meets the benchmarks of traditional FPGAs but optimizes processing speed and power efficiency.
- **Reconfigurable Resources:** Establishing a reconfigurable framework capable of adjusting the allocation of core FPGA components.

The chapter was structured in two parts: initially, the system architecture of our design was presented, providing an overarching architectural overview and explaining the 'unified tile' concept. Subsequently, detailed implementations for the components used in the tile were explained. This implementation approach was crucial to validate the feasibility of the components and establish a foundational understanding for the electrical consumption and delay analyses in upcoming chapters.
While three of the four design objectives have been addressed in this chapter, the performance improvement objective will be discussed in the upcoming sections of this thesis.
Addressing the design objectives, our architectural proposal showcased a capacity to reconfigure the core functionalities – memory, interconnect, and logic – at a system-wide level. The high-level architecture consists of a 2D grid of homogeneous tiles, each of which can be reconfigured to perform any core functionalities. This design ensures flexibility, unhindered by placement/routing constraints, permitting resource optimization per application requirements.
The core of the architecture is the 'unified tile'. Within each tile lies the crossbar array, tasked with storing logical functions, routing configuration or just data for storage, depending on the tile's operational mode. The tile features an internal routing system, specifying data input sources and directing output destinations to neighboring tiles. This combined with peripheral circuitry to control the crossbar array, describes the composition of each homogenous tile. Note that the differentiation between operational modes is determined solely by the content of the memory array and the routing of input connections to components in the tile.
Further, we detailed potential electrical circuits vital for realising the unified tile's components. The crossbar array, inherently designed for eNVM technologies, was for this thesis structured around FeFET. While ReRAM can be directly dropped into the place of the FeFET, other technologies, such as STT-MTJ and SRAM, can not be implemented for specific components. The crossbar array consists of interconnected memory cells accessed via an access transistor. The connectivity is regulated by selection circuits, which interface with the crossbar array's columns and rows. A dedicated sense amplifier, linked to each column, determines the state of the selected memory cell.
In conclusion, this architecture satisfies the objective of surpassing a mere SRAM replacement of common FPGA components, introducing a unified component adept at performing all three core functionalities. Furthermore, the design prominently relies on eNVM technologies for its key components.

5

Method of Verification and Testing

In Section 5.1, the overarching framework of the testing method is presented. This method is built on two key components: a behavioural and an analytical model for the unified tile. As detailed in Section 5.2, the behavioural model is capable of accurately generating the output of a tile. The analytical model, discussed in Section 5.3, is used to evaluate the electrical performance of the system.
Section 5.4 elaborates on our comparative methodology. This approach has two objectives: the first is to analyse the performance of various memory technologies, and the second is to compare our design against state-of-the-art architectures. Section 5.4.1 introduces a comparison framework to aid in these comparative analyses, ensuring a structured evaluation of differing architectures. Section 5.4.2 presents the metrics used to analyse the electrical performance of the architecture.
These sections lead to a complete framework to test the behaviour and performance of this architecture. In Section 5.5, we present two micro-benchmarks we utilise to test various aspects of the architecture.

5.1. Testing method

The testing method outlined in this section, is driven by two goals. First, it seeks to validate the functional correctness based on the specifications as presented in Section 4.2. Second, it extrapolates the electrical performance of the system based on the circuits presented in Section 4.3. Figure 5.1 offers a visual representation of the structure of the testing method.
Emerging eNVM technologies poses some challenge regarding design testing. Traditional digital designs, described with HDL languages like VHDL, are commonly verified on platforms like FPGA. However, given the use of emerging eNVM in our architecture, such a standard testing method can not provide accurate results on the electrical performance.
For the generations of results, it is not computationally efficient to use a circuit-level simulator. We use a custom build event-based simulator implementing a behavioural and analytical model of the system, which implement a accurate portrayal of behaviour of emerging eNVMs. The behavioral model is capable of simulating the output of a tile based on a given input, allowing for the analysis of signal propagation and generation in the system. Furthermore, The addition of a timing models ensures that thes signal propagation is cycle-accurate.
Electrical performance is determined by a second model, we call the analytical model. The parameters of this model are determined by a combination of SPICE circuit-level simulations and literature-backed specifications. By combining results from the behavioural modal, we can accurately extrapolate the dynamic behaviour of a tile. The dynamic behaviour is used to extrapolate the electrical performance.

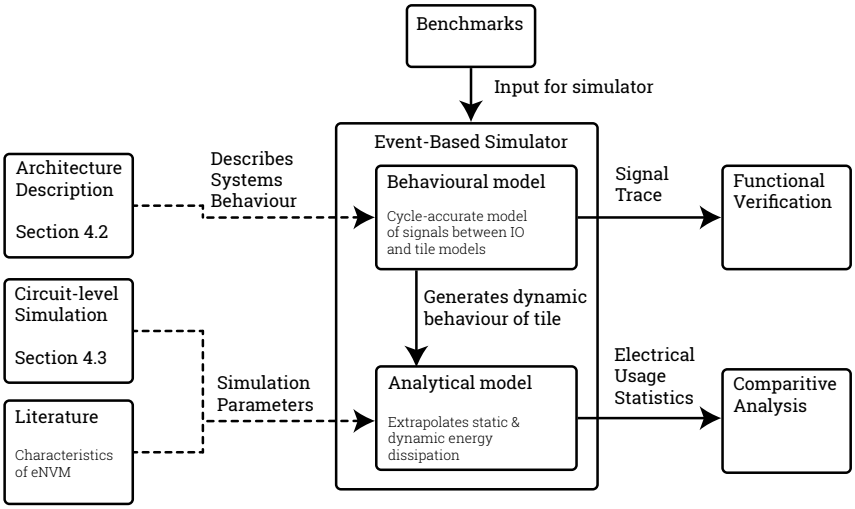


Figure 5.1: Overview of Testing Methodology

5.1.1. Limits of the Testing Method

A key aspect of this research is an comparative study of this architecture against other architectures, which would involves the use of benchmarks. Several sophisticated benchmarks have been developed to test new FPGA designs. Those benchmarks test different aspects of an FPGA, for instance, the impact of modifying the size of the LUTs in the logic blocks or adjusting the number of CLBs and their relative placement to other CLBs and switch blocks.
An older yet commonly applied benchmark is the MCNC20 [27], comprising of set of logic-centric applications with few registers. Due to their limited size, these benchmarks may not produce representative results for modern FPGAs. The more recent Titan benchmark [28] offer a more appropriate basis for comparison with modern FPGAs. It is worth noting that several studies in the literature continue to use the MCNC20 benchmarks when comparing architectures.
Relying on these benchmarks to compare this architecture with its counterparts is, for this thesis, impractical. This limitation stems from the fact that our focus has solely been on developing an architecture and modelling the specific components. Mapping these benchmarks to this architectures manually is near impossible due to the complexity; Furthermore, synthesis tools which can reconfigure resources like (interconnect, logic, memory) do not exists, most of these tool can only operate when the resource topology is predefined. In other words a tool capable of synthesizing the Titan benchmarks to this architecture has yet to be developed. An open-source tool like Verilog to Routing (VTR) [29] would provide an foundation for such a tool. However, substantial research is still required to change a tool like VTR to synthesize system descriptions for this architecture.
Due to the absence of a dedicated computer tool, we utilize micro-benchmarks which are manually converted to systems descriptions, to demonstrate some capabilities of the tile architecture. This method using micro-benchmarks is comparable to the comparison method used in [23].

5.1.2. Simulation with SystemC

When simulating system behaviour, various methods are available; one such way is based on RTL waveform simulation using hardware description languages. Amongst them, Verilog and VHDL are the most prominent. Our primary interest lies in the latter, commonly utilized for both design and verification. For both languages, a broad spectrum of simulators is available to execute in RTL simulation.
A custom simulator can accurately represent system behaviour and quantify electrical usage, considering the specificities of emerging eNVMs. In this context, based on C++, the SystemC library offers a fitting solution for this project. SystemC consists of classes specifically designed for high-level behavioural modelling. Its C++ foundation encapsulates model descriptions as C++ functions, enabling more straightforward integration with the analytical model to generate power metrics.
In this research, we employ the Accelera SystemC library [30], based on the IEEE 1666-2011 standard [31].

5.2. Behavioral Model

We developed a behavioural model of the unified tile architecture to enable a functional verification and analysis of the system's dynamics. This verification architecture simulates a system composed of several interconnected, unified tiles, implementing the 2D matrix structure where each tile operates in parallel. The testing environment, illustrated in Figure 5.2, is implemented with the SystemC framework. It comprises a system definition, an NxM matrix of interconnected tiles based on user input, and an input stimuli description.
Consistent with the specifications from Section 4.2, tiles interface only through the four dedicated channels linking them to their neighbouring tiles. The tiles on the matrix's boundary (on the first or last rows/-columns) can connect with a driver or a monitor. Based on user specifications, the driver drives signal sequences to the connected tile while the monitor observes and records the tile-generated outputs.
To mirror the system's constraints, we have restricted the input of signals to only tiles at the matrix's periphery. All the inter-tile communication channels are monitored to enable post-simulation analysis of signals. All the generated signals by tiles are exported into VCD files, which can be read with conventional waveform visualization tools.

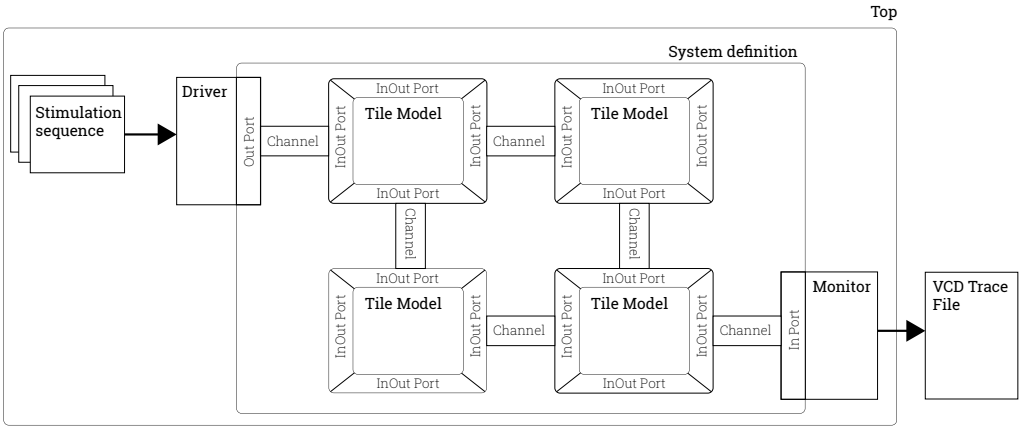


Figure 5.2: Overview of Verification Architecture

5.2.1. Model of the Unified Tile

At the core of the simulation architecture lies the model of the unified tile, which consists of several components that interact with each other. Figure 5.3 gives an overview of these components. The figure shows the primary components used to simulate the tile's behaviour and performance accurately. As explained in previous sections, the simulator must be able to generate cycle-accurate results based on the state of the tile, the content of the memory array and the input given to the tile. Each tile consists of 5 inputs; one is a clock, used to synchronize sequential behaviour, which we will further discuss later in this section. The four other signals represent the tile's connections with adjacent tiles, a monitor or a driver. The tile's position in the matrix intrinsically determines the connections a tile has. The channels between tiles can be written to and be read from by the connected tiles. The signals on these channels can be either logical 1 or 0 based on the model's results or undefined ('U') when the tile does not write the channel. Because there is no separate read or write path between tiles, it is possible to have two tiles write the same channel simultaneously. This behaviour will make it impossible to determine the state of the channel to accurately describe this a fourth state is being used called metastability ('X').
Therefore, the input for a tile is 4 N-bit channels. A tile can only process a N-bit value. A direction is selected to compose this N-bit value. In Section 4.2, we explained that it is possible to choose the direction (North, East, South or West) at bit-level. So, the signals given to its inputs are filtered to a single N-bit signal. This filter is called the "direction manager" (DM). The DM determines the bit flow to and from the four directions.
An N-bit output is generated based on the state of the tile, which consists of multiple parameters. We define the following parameters for each tile:

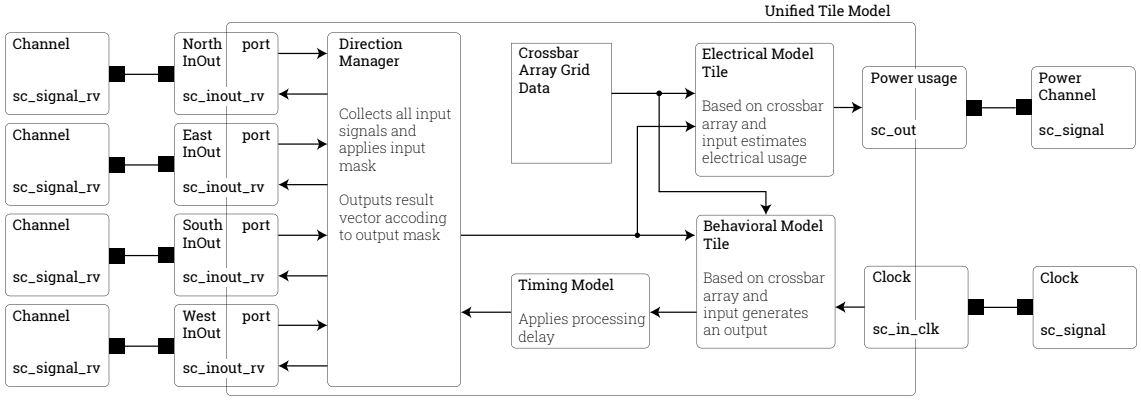


Figure 5.3: Unified Tile Model Structure

- **Size of Tile:** Each tile has a size $N \times M$ where N is the width of the crossbar array, and M is the height. This size determines the amount of data saved in the crossbar array and the number of connections between each tile.
- **Mode:** If the tile is configured in either logic, interconnect or memory mode.
- **Crossbar array data:** The data retained in the crossbar array can be set pre-simulation.
- **Connections:** Each tile can be configured for each bit in the four directions if it is an input or an output or not used.
- **Through routing:** For each bit in each direction, it can be determined if it is directly routed through without going through the crossbar array.
- **Flip-flop configuration:** Determines if an SA is connected to a flip-flop or directly connected to the output

The mode of a tile is its most influential parameter, as it most dominantly determines the output. Section 4.2 presents a detailed discussion about the expected behaviour of the various modes. Essentially, the mode dictates how and which information is selected from the crossbar array. For this reason, the data in the crossbar array is another critical determinant of a tile's response to an input. Combining the data in the crossbar array, the mode, and the input collectively defines the output of all logical operations performed within a tile. If an output is not directly linked to an SA but rather to a flip-flop, the clock influences the signal at the associated output of the tile. This is because the specific bit will only change at the clock's flank; otherwise, the outputted value remains as in the previous clock cycle.
Two other parameters define a tile's routing behaviour. The connection masks determine which bits are selected from which direction, ultimately influencing the input to an operation inside the tile. A neighbouring tile can also connect to the input of another tile, especially when multiple tiles need the same input. Essentially, a tile can duplicate the signal from the input to the output, thus routing signals through the tile's periphery rather than over the crossbar array.
An example of this behaviour is illustrated in figure 5.4. Here, we receive four inputs from four directions: bits 1 and 0 from the north, bit 2 from the west, and bit 3 from the south, resulting in the input signal 1011. The tile, set in logical mode and sized 4x4, interprets the signal 1011 to select column 2 and row 3 ('10' = 2 and '11' = 3). Based on the data within the crossbar array, this produces a 0. However, since we have set the output to connect not directly to the SA's output but to a flip-flop, we obtain the results from the prior clock cycle. As the clock signal transitions from low to high, the output changes to the result determined by the provided input, which is '0'. We have configured the tile so that results are sent in the east direction. Consequently, the neighbouring tile in the east will receive a '1' at bit 3 from this tile.

Timing model SystemC is a discrete event-based simulator, allowing for precise simulation of the time an events occur. Sequentially arranged tiles propagate signals from one tile to the next until they reach the output. Given that the architecture is underpinned by electrical circuits, primarily reliant on CMOS technology, information doesn't propagate instantaneously. Instead, it incurs delays caused by

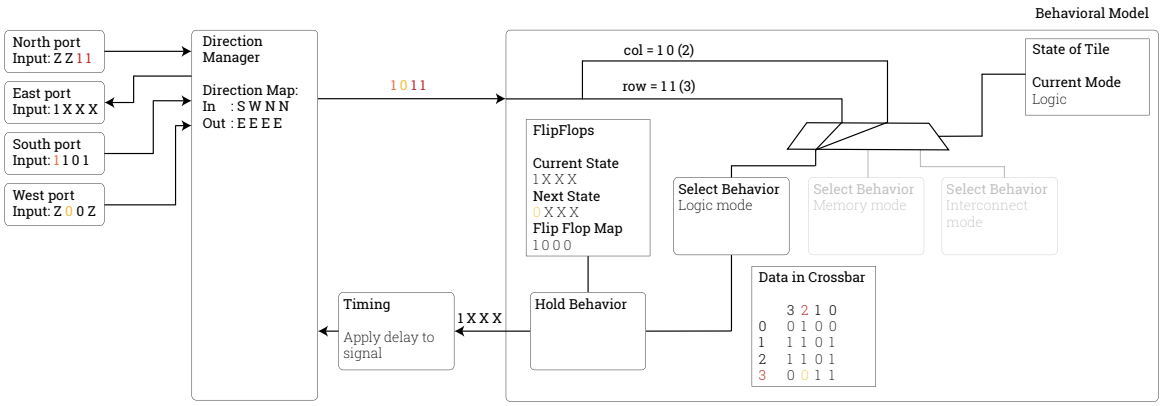


Figure 5.4: Direction Manager Operation Example

the switching of various design components. By incorporating these delays, we can accurately produce cycle-accurate signals. In this thesis, our focus narrows to the most crucial operations impacting overall performance. These include:

- **Sensing Time:** The duration required for the sense amplifier to ascertain the FeFET's state following the excitation of the word and bit lines.
- **Activation Time:** The lag between an input directed at a tile and the excitation of the word and bit lines, essentially the propagation delay from the external periphery to the selection circuits.
- **Programming Delay:** The time taken for the ferro-electric layer of the FeFET to switch when set in memory mode.

We do not delve deeper into the delays needed for programming individual tiles during their reconfiguration phases.
With the single-ended latch sense amplifier, we've observed a sensing time of approximately 50ps for the 90nm technology node at $V_{DD} = 3.3V$. The corresponding circuit is showcased in Figure 4.18a in Section 4.3.3. A graphical representation of the output rise time is available in Figure 5.5.

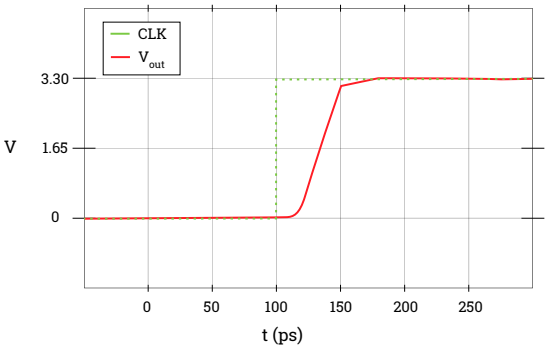


Figure 5.5: Sensing time of circuit in figure 4.18a

This delay, in conjunction with the clock cycle, determines whether sequentially arranged tiles can produce accurate results at a given clock frequency. If the clock frequency is too high, the generated signal may exceeds timing requirements, potentially leading to metastability.

5.2.2. Method of Verification

In Section 5.5, we presented two micro-benchmarks capable of testing several aspects of the architecture. It is well-understood that a sequence of lookup tables when paired with flip-flops for storage, can emulate any digital logic operation, as proved by the foundational elements in an FPGA's CLB. Thus, we can assume these benchmarks can be mapped on this architecture. However, we must ensure that what we mapped is correct. Thus, we need to execute a functional verification.

We have adopted a directed testing methodology to ensure the mapped applications' functional integrity. Directed testing implies that we target specific functionalities or configurations. With a few manually constructed test cases to stress or validate specific aspects of the architecture. Post-simulation manual inspection of the waveforms is used to determine if the expected results are produced.

Verifying the design of an adder is straightforward. If we want to test if an adder is functional correct, we provide 2 N-bit integers. To prove if the adder is working correctly, we check the output of the adder to confirm that it is the sum of the provided 2 N-bit integers.

For the load-store architecture, it is less trivial to reach 100% test coverage. Producing a test case for every state the system can be in is impractical. So, we first test if the individual components produce the expected results. For instance, we can test if four connected tiles in memory mode fetch the correct row from one of the four tiles. By confirming the correct behaviour of individual components, we can more confidently assemble every component to a form the system. We do a final run of a set of instructions and confirm if the fully assembled systems still produce the expected results.

In summary, our testing method involves directed testing of carefully constructed test cases. By manually checking the waveforms produced by the simulator, we confirm the functional correctness of the mapped applications.

5.2.3. Summary

The behavioural model explained in this section offers a precise representation of the behaviour of a unified tile. The behavioural model is capable of determining the output of tile based on the state of the tile, its memory array's content, and inputs. Combined with a timing model we can effectively simulate cycle-accurate communications between tiles and IO.

Central to the testing method, this model is essential for two primary goals. First, it facilitates the functional verification of a mapped program. The model can determine if a program is executed as expected by simulating signals between tiles and IO. Second, the model provides insights into the dynamics of the tiles. These dynamics are used to determine the switching behaviour inside of a tile. In the next section, we discuss how the these dynamics serve as an input to the analytical model, which is capable of determining a tile's static and dynamic energy usage.

5.3. Analytical Model

In the previous chapter, we introduced the behavioural model of the unified tile. The behavioural accurately represents the transmitted signals on the tiles and IO interconnections. The signals on the interconnect between tiles are effectively a tile's input and outputs. In this Section, we will discuss how we model the electrical behaviour of the tile system based on the results obtained from the behavioural model.
The electrical behaviour of a system is linked to the implemented circuits. In Section 4.3 we presented a implementation of the main components of the tile. We will consider these implementations for our analysis of the electrical behaviour of the tile system.
The analytical model considers key performance metrics for evaluating overall system power usage. An overview of the given metrics is given in the following list:

- **Power Consumption:** Quantifies the circuit's energy dissipation during the execution of a program, which consists of:
  - **Static Power:** The energy consumption in the attributed to the leakage current of the used components
  - **Dynamic Power:** The energy dissipation caused by the switching of the transistors.
- **Circuit Area:** Denotes the area a design occupies on the chip.

The following Section presents the device models used for our SPICE simulation. These models, in combination with results from similar components, are used to determine the device characteristics. In Section 5.4.1, we present the methods used by the simulator to extrapolate static and dynamic energy power usage from the SPICE simulations with the device models.

5.3.1. Device Models

This section explores the different device models used in SPICE simulations to estimate the electrical characteristics of the implementation used in this architecture.
Central to this thesis are these device models:

- **NMOS and PMOS:** These MOSFETs constitute most CMOS structures used in the periphery and sensing circuits. The BSIM4 model applied at a 90nm technology node simulates the behaviour of these MOSFETs.
- **FeFET:** The FeFET is modelled based on its associated drain-to-source resistance in its current state. Resistance values are derived from the literature.
- **ReRAM:** The ReRAM is modelled as a resistance determined by its current state. Resistance values are derived from the literature.
- **STT-MRAM:** The STT-MRAM is modelled by placing a resistance in series with a BSIM4 model. Resistance values are derived from the literature.

In the subsequent paragraphs, we will further discuss the used device models, mainly focusing on the method used to determine specific characteristics associated with different states of the emerging eNVMs. An explanation of the devices' programming and the representation of this behaviour within the model will also be provided.

MOSFET Model

The BSIM (Berkeley Short-Channel IGFET Model) models are mathematical models for MOSFET behaviour. Developed by the University of California, Berkeley [32], these models are commonly used in IC design and electronic design automation due to their accuracy in predicting transistor behaviour across various operating conditions.

Baseline SRAM Model

To compare the different memory technologies, we use the following works [33, 34]. The difference between these and the presented circuits, which will be used for the eNVMs, is that for the SRAM, we use the more common differential sensing circuit, like the one we presented in figure 4.17. In Table 5.1 are the electrical characteristics of the SRAM cell and the Differential sensing amplifier. Both results are for the 90nm technology node.

Table 5.1: Electrical Characteristics SRAM and sensing from [33, 34]

Frequency	P Memory Cell (μW)	P Diff. SA (μW)
500 MHz	4.890	188.9
1 GHz	8.002	259.6
2 GHz	11.979	388.6

eNVM Models

FeFET In addition to the CMOS structures employed, FeFET devices are also utilised in the presented design. The power consumption of these devices is determined by the state of the ferroelectric layer, the voltage across the drain to the source, and the applied gate voltage. Figure 5.6 illustrates the resulting drain-to-source current, adapted from [10]. To accurately model the circuit behaviour, it is necessary to incorporate the depicted power consumption into the complete model of the crossbar array.

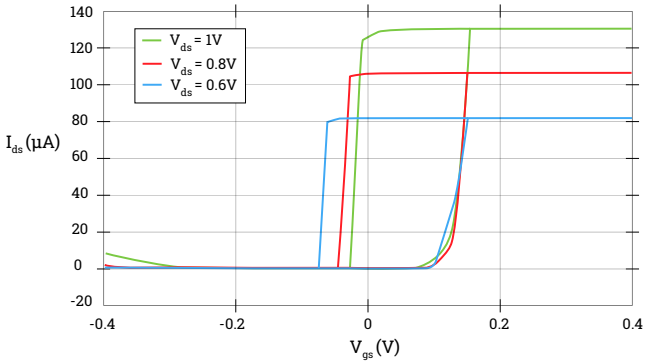


Figure 5.6: Line graph of the hysteresis loop of the FeFET device from [10]

The behaviour of interest comprises the following parameters:

- **Power Consumption during Sensing:** A FeFET can be in three distinct states, influencing the amount of current over the drain to the source.
  - *MOSFET OFF:* The gate voltage is below the threshold, making it impossible to evaluate the state of the FeFET, and a logical 0 is read.
  - *MOSFET ON, FE-layer state is 1:* The gate voltage is sufficiently high to evaluate the state of the ferroelectric layer. In this case, the layer is polarised in the '1' state, resulting in a relatively low resistance between the drain and the source.
  - *MOSFET ON, FE-layer state is 0:* The gate voltage is sufficiently high to evaluate the state of the ferroelectric layer. In this case, the layer is polarised in the '0' state, resulting in a high resistance between the drain and the source.
- **Power Consumption during Programming:** Changing the state of the ferroelectric layer requires a different amount of energy than evaluating the state. These values can be obtained from the literature.
- **Programming Delay:** Changing the state of the ferroelectric layer relative to the voltage applied to the gate requires a certain amount of energy, leading to a specific programming delay. These values will be based on the literature.

The graph presented in Figure 5.6 determines the power consumption during sensing. These values are for a FeFET device with a feature size of 90nm. If the state of the FeFET is ON and a $V_{DS} = 1V$, then a $I_D = 130\mu A$ is observed. Using Equation 5.1, the $R_{DS}(ON)$ can be determined. For this example, this results in a $R_{DS}(ON) = 7.7k\Omega$. These values can be used to determine the power consumption of the FeFET during sensing.

$$R_{DS}(ON) = \frac{V_{DS}}{I_D} \tag{5.1}$$

Literature provides the programming delay and programming energy required to switch a FeFET device. Chen et al. simulated the circuit-compatible model of [9] in [17], obtaining a programming delay of approximately $1\mu S$ and an energy loss of $57.11fJ$.

ReRAM ReRAM devices are two-terminal devices that do not separate their programming and reading paths. A ReRAM device can, therefore, be modelled as a resistance whose value is determined by its state. Although research indicates that ReRAM can hold multiple states besides ON and OFF, this work only considers two states for the ReRAM device.
The behaviour of interest includes the following electrical parameters:

- **Power Consumption during Sensing:** The ReRAM device is either ON or OFF. The resistance value and, consequently, the losses over the device depend on the state.
- **Power Consumption during Programming:** To change the state of the ReRAM device, a specific voltage and current must be applied, resulting in associated power consumption.
- **Programming Delay:** Programming these devices takes longer than evaluating their state, resulting in a programming delay.

The associated values are an ON resistance of $10K\Omega$ and an OFF resistance of $1M\Omega$ obtained from [35]. The programming energy of the ReRAM device is estimated by the relation given in Equation 5.2. The programming delay of the ReRAM device is $10nS$ [6].

$$P_{program} = \frac{V_W^2}{\sqrt{R_{ON}R_{OFF}}} \tag{5.2}$$

5.4. Methodology for Comparative Analysis

In the previous sections, we established device models we will utilise in the SPICE circuit-level simulations. These results with literature data form the simulation parameters used to extrapolate key electrical metrics for the benchmarks. This section provides a comprehensive overview of the method used by the analytical model to extrapolate these electrical characteristics. In the subsequent discussions, we detail our approach to comparing new eNVM-based RCAs, analyse power dissipation, how the analytical models generates these results and highlight the important considerations for assessing system performance.

5.4.1. Key metrics for emerging eNVM-based architectures

In Section 2.3, we explained that the power dissipation in CMOS circuits is divided into two parts: static energy consumption and dynamic energy consumption. Comparing different technologies like CMOS ReRAM or FeFET with each other is challenging because these technologies are fundamentally different. In Section 2.3, we discussed how we can determine electrical performance metrics for CMOS circuits. This Section will discuss how to use those concepts to compare different architectures based on these emerging eNVMs. The presented work is based on the work of Reuben et al. in [36].

In our architecture, several distinct components primarily contribute to the electrical characteristics. Based on the architecture presented in Section 4.2, we categorise four fundamental components present in each tile:

- **Selection Circuits:** These circuits, explored further in Section 4.3.2, decode a input signals into specific activation of the WL and BL lines.
- **Sensing Circuits:** Detailed in Section 4.3.3, these circuits are tasked with determining the state of the eNVMs.
- **Memory Elements:** Used to store information, in our case the components in the crossbar array.
- **Input/Output Logic:** This encompasses additional components, like flip-flops, that augment the functionality of the tile.

As highlighted earlier, we can employ a basic analytical approach to determine the majority of energy used by CMOS devices. However, adopting the BSIM model and SPICE simulations allows us to achieve more precise results. These give deeper understanding into the switching behaviour of the CMOS circuits, enabling a distinct separation between static and dynamic energy consumption.

Energy dissipation in memory array
The components detailed here are not exclusive to our design. Instead, they are typical systems based on the emerging eNVM technology. It is crucial to identify these components when evaluating the electrical performance of such systems. This is due to most implementations' consistent structural pattern, as evidenced in works like [37, 38]. Reuben et al. identified three main components of interest:

- **Memory Array**: Which are the memory elements and their lines
- **Peripheral circuits** comprise the architecture of this thesis of the sensing circuits, the input/output logic and the selection circuits needed to drive the word- and bit-lines.
- **Controlling circuits**: Is auxiliary hardware needed to control data flow and signals through a tile. Like a finite state machine which controls sequential operations.

For each component, it is possible to outline their electrical specifications formally. We will begin by defining the energy dissipation of the memory array. The subsequent relation characterises this dissipation:

$$E_{array} = E^{A}_{dynamic} + E^{A}_{static} \tag{5.3}$$

The dynamic energy of circuits emerging eNVMs is comprised of two components: these are - similar to CMOS technology - comprised of the energy spent on switching the devices. Moreover, the energy spent to charge the capacitive load of the lines in the memory array. We define the average energy spent for switching an emerging eNVM as E_{sw} and the total capacitive load of the array as E_{cap}. These components combined give the relation for the Dynamic energy spent in the memory array presented in formula 5.4 where n is the number of memristive devices in the memory array.

$$E_{dynamic} = n \cdot E_{sw} + E_{cap} \tag{5.4}$$

The static energy of the memory array, as defined by Reuben et al., is also defined by two components. These are the energy dissipation of all non-switching emerging eNVMs defined as $E_{leakage}$. The second component is the residual static energy dissipated after an emerging eNVM switch, defined as $E_{residue}$. Giving the following relation.

$$E_{static} = E_{leakage} + E_{residue} \tag{5.5}$$

Energy dissipation in controlling and peripheral circuits
The second major component defining the energy dissipation of this design and designs using emerging eNVMs is the circuitry needed to excite the word- and bit-lines to compute the circuits. The structure of the memory array can be estimated with the previously presented equations; these apply to most of the common implementations. However, how data is moved to and from the memory array is implemented differently over different kinds of architecture. In [36], a comprehensive discussion is given over these differences. This thesis will concentrate on the implementation based on near-memory processing. Because these closely represent the developed architecture.
These implementations rely on three components: circuitry for sensing the state of the emerging eNVMs device, controlling the word- and bit-lines and additional circuitry to control the data flow. SA are commonly used to determine this state, and the energy consumed can be estimated by the following relation presented in 5.6. E_{sa} is the energy consumed for reading a single bit, and E_w is the energy dissipated for changing a single bit. $N_b(i)$ are the number of bits being read and written to during an operation.

$$E_{read} = E_{sa} \cdot \sum_{i=1}^{N_{read}} N_b(i);\ E_{write} = E_w \cdot \sum_{i=1}^{N_{write}} N_b(i) \tag{5.6}$$

We define the two components for controlling circuits as $E_{controller}$ and for periphery circuitry as $E_{peripheral}$. These components' energy dissipation is proportional to the processing steps needed to reach a computational result. We define these computational steps as N_P. For the sensing/peripheral circuits, we get the following relation.

$$E_{peripheral} = E_{read} + E_{write} + N_P \cdot E_P + E^{P}_{static} \tag{5.7}$$

Where E_P represent the energy dissipated in used circuitry after the sensing amplifier like transmission gates or multiplexers, the energy dissipated by the controller is defined by the switching energy of the circuitry used. The total energy spent is proportional to the number of reading steps N_{read}, the number of writing steps N_{write}. N_{CS} are the computational steps and are a measure of complexity to execute a specific function in the architecture. N_{read} is a function N_{CS} because the sequential steps needed to execute a function directly influence the number of read/write operations.

$$E_{controller} \propto (N_{read}(N_{CS}) + N_{write}(N_{CS})) \tag{5.8}$$

This leads to the total energy dissipated in the system, defined by the following relation.

$$E_{total} = E_{array} + E_{peripheral} + E_{controller} \tag{5.9}$$

Estimation power usage in analytical model
The previous section gives us a framework to evaluate the electrical performance of a tile. This section provides a comprehensive overview of the method used by the analytical model to extrapolate these electrical characteristics based on this framework.

Static Power The analytical model considers static energy usage of the memory array represented as E_{array}, SA, selection, and input/output components which combined represent the $E_{peripheral}$. We can determine each individual component's static energy usage based on the circuit-level simulations with the models presented in Section 5.3.1 and from characteristics found in literature.
We then extrapolate the static energy usage to the amount of tiles and the size of the tiles to estimate the total static energy usage. This gives us the following relation to determine the system's power usage. Where n_{system} is the total number of tiles used for an application. N is the number of rows in a tile, and M is the number of columns. E_{leak} represents the static energy dissipated from the different components.

$$E^{P}_{static} = \sum^{n_{system}} E^{tile}_{leak} \tag{5.10}$$

$$E^{tile}_{leak} = \sum^{N} \left(E^{In}_{leak} + E^{rowsel}_{leak}\right) + \sum^{M} \left(E^{SA}_{leak} + E^{colsel}_{leak}\right) \tag{5.11}$$

$$E^{A}_{static} = \sum^{N \cdot M} E^{memcell}_{leak} \tag{5.12}$$

E^{In}_{leak}, E^{SA}_{leak} and $E^{memcell}_{leak}$ are independent from the size of the tile. Thus, more columns will result in more static energy lost on sense amplifiers. However, this increase is caused by adding individual sense amplifiers, not an increase in static energy usage for each sense amplifier. E^{colsel}_{leak} and E^{rowsel}_{leak} represents the individual amount of static energy dissipated for each additional row/column. The individual components behind each row/column will also increase because more transistors are needed to implement a bigger AND gate.
In conclusion, static energy in CMOS circuits is closely tied to the number and size of tiles used. While components like sense amplifiers and memory cells have consistent energy costs irrespective of tile size, adding rows or columns increases components' number and complexity, increasing static energy dissipation.

Dynamic energy The behavioural model gives for a given application accurate on the inter-tile communication. We can determine from these communications between tiles the input to an individual tile. Based on the tile's mode, the memory array's content and the flip-flops' state, we can accurately determine the amount of transistor switching occurring in a tile. We can accurately estimate the dynamic energy of the whole system by the sum of all estimated switching behaviour for each tile.

$$E_{Dynamic} = \sum_{i=1}^{N_{system}} E_{Dynamic}(tile)(i) \tag{5.13}$$

We consider two types of read operations for dynamic energy: reading a one and reading a 0. The sum of the read operations performed to generate the tile output represents all dynamic energy dissipated in the grid and the SA combined. These values are obtained with SPICE simulations with the models presented in Section 5.3.1 and results from the literature, which are then extrapolated by the simulator. All results are checked against similar implementations in the literature. We consider the average amount of energy dissipated for rewriting the memory cell and multiply it by the amount of write operations performed. Furthermore, we take the average switching energy associated with the selection for a change in row/column selection. Finally, we add the average dynamic energy spent for the circuits at the output.

$$E_{Dynamic}(tile) = E_{read} + E_{write} + N \cdot E_{sw} + E_{output} \tag{5.14}$$

We consider an accurate number of read and write operations from the behavioural modes. These models combined gives us a accurate representation of the dynamic energy dissipation.

Latency
The latency of these systems is defined by the time it takes to move data from the controller to the memory array to the peripheral circuitry. We define these characteristics: T_{read}, T_{write} respectively, and the time it takes to read and write to a memory cell. T_P is the delay caused by processing steps after the sensing action. T_{route} is the time it takes for information to move between components specific to a particular architecture. These values determine the total latency for an operation consisting of multiple computational steps to generate a result.

$$T_{compute} = N_{CS} \cdot T_{route} + N_{read}(N_{CS}) \cdot T_{read} + N_{write}(N_{CS}) \cdot T_{write} + N_P(N_{CS}) \cdot T_P \tag{5.15}$$

The computational steps are defined separately from the actual delays. This gives us a merit to compare designs using computational steps. Independently of, used sensing circuits.

Area
The total area of systems using emerging eNVMs combines these components and CMOS circuits. These eNVMs can be used in various components, which will be further explored in this chapter. The total area on-chip will be set after production, but an exciting merit is the effective area utilisation for an application. The following relation can define this.

$$A = A_{cells} + A_P + A_C \tag{5.16}$$

A_{cells} are the participating memory elements for specific applications. A_P and A_C are the periphery circuitry and the controlling circuitry, respectively, used to control the participating memory elements.

Summary
In this Section, we presented the relevant metrics needed to determine the electrical specifications of the system. The simulator calculates the electrical parameters to give insights into the operational power usage of different designs. We can use the other metrics to compare other state-of-the-art architectures. The following table presents the key metrics used to make this comparison.

Table 5.2: Overview key metrics comparison emerging eNVMs based systems

Metric	Formula
Energy	$E_{array} + E_{peripheral} + E_{controller}$
Latency	$N_{CS} \cdot T_{route} + N_{read}(N_{CS}) \cdot T_{read} + N_{write}(N_{CS}) \cdot T_{write} + N_P(N_{CS}) \cdot T_P$
Area	$A_{cells} + A_P + A_C$

5.4.2. Comparative Analysis

The previous sections have detailed our system's architectural design's underpinnings and energy-related considerations of emerging eNVMs-based systems. This comprehensive understanding sets the stage for comparing different memory technologies and other state-of-the-art architectures.
While our simulator offers insights into the power and delay metrics within the unified tile's context, it becomes pertinent to interpret these metrics in a broader scope. This allows us to discern which memory technology is best suited for specific applications and which offers the most efficient system-level performance.

Efficiency Metrics

We employ two benchmarks: one is purely logic-based, and the other is a more complex design with multiple aspects to it. Only the first benchmark will be compared with other state-of-the-art architectures because those utilized a similar benchmark in [23]. Nevertheless, in our comparison of the different memory technologies, we analyse the results of both benchmarks. In our analysis, we will use the following efficiency metrics.

- **Joule per Operation**: This will provide a clear picture of how many operations can be executed with a unit of energy. It is a critical metric to understand the cost-effectiveness of each memory technology.
- **Power-Delay Product (PDP)**: PDP quantifies the balance between power consumption and its operational speed. A lower PDP suggests a more efficient design, indicating reduced power use for a given performance or quicker operations for the same energy.
- **Energy-Delay Product (EDP)**: EDP captures the trade-off between speed (delay) and energy efficiency. A lower EDP indicates a more optimal balance between these two metrics.

5.4.3. Conclusion

This methodology is a framework for examining the unified tile architecture's performance across diverse memory technologies by employing a simulation-based approach using micro-benchmarks.
To get a comprehensive insight into this architecture, we will compare this architecture on two fronts. Firstly, we aim to determine the optimal memory technology for the unified tile architecture by comparing energy efficiency, operational speed, and balance. Secondly, we will compare our architecture with other state-of-the-art architectures. This analysis will use the energy consumption, delay, and efficiency metrics.

5.5. Micro-Benchmarks

The preceding sections provided an overview of the simulations, models, and methods employed to ascertain the electrical characteristics of the system. This section details the applications utilized to validate the implementability of a circuit, and to measure the resources and power consumption. These applications are referred to as benchmarks because they serve to evaluate the performance of various memory technologies. For this thesis, three benchmarks were developed to demonstrate how applications are mapped onto the architecture. These benchmarks are as follows:

1. **Ripple-Carry Adder:** A ripple-carry adder is a straightforward adder that can process two N-bit binary numbers and return an (N+1)-bit binary number, which is the sum of the two binary numbers.
2. **Load-Store Architecture:** A load-store algorithm segregates ALU instructions from data. Consequently, memory access is employed to fetch an instruction and data, then an operation on that data is performed, and the result is subsequently stored back into the memory.

5.5.1. Ripple-Carry Adder

An adder is the fundamental element of numerous arithmetic operations in most computer applications. The ripple-carry adder is one of the most straightforward variations of an adder. It comprises a series of single-bit full adders, wherein the carry bit transmits from the least significant bit to the most significant bit. Consequently, the delay of this adder has a linear relation with the width of the input bits.
At its most basic, an adder is composed of two logic gates: an XOR gate, which determines the result, and an AND gate, which determines the value of the carry bit. This configuration is known as a half-adder. A full adder, on the other hand, can process two bits and a carry bit. It consists of two half-adders and an OR gate, which determines the result of the carry bit. A sequence of these full adders can sum any two N-bit binary values. Figure 5.8 illustrates both types of adders, and Figure 5.7 illustrates how multiple single bit-adders form a ripple-carry adder.

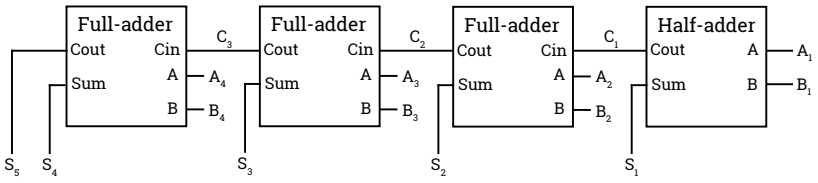


Figure 5.7: Ripple-Carry Adder

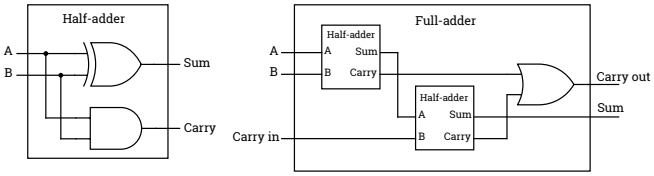


Figure 5.8: Logic of half- and full-adder

There are several ways to configure an adder. For example, we show a 4-bit adder like the one in Figure 5.7. The design space consists of many parameters. In the following example, we use a tile size of 8x8; thus, every tile consists of 64 bits, translating to a 6-input LUT. For the ripple-carry adder, we only consider the logic part, thus not the routing to and from the tiles. There are several possible configurations when using an 8x8 memory array. Figure 5.9 is the specific configuration depicted, which we will simulate.
Figure 5.9 shows the same structure as the ripple carry adder from Figure 5.7. The least significant bit enters the tile on the right, and then the carry signal propagates to the left. The adder generates the resulting signals sum and carry in parallel. Therefore, the sensing delay of a tile is the time it takes to generate the result of a single bit. In table 5.3, the truth table defines the adder's combinatorial logic.

The total time to generate a result for a 4-bit integer is four times this sensing delay of the tile. The total delay is linearly related to the number of bits of the summed integers.

Table 5.3: Truth Table of the Full-Adder

Cin	B	A	Sum	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

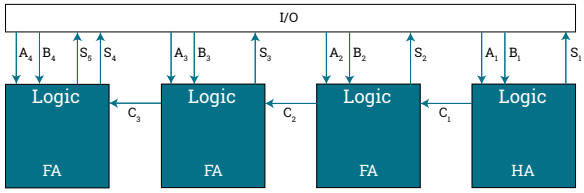


Figure 5.9: Benchmark Ripple Carry Adder

Improved Adder When we do not strictly conform to the structure of a ripple carry adder, a faster and higher efficiency of the memory grid is expected. For the previous example, every tile consists of a single-bit full adder. The underlying hardware is set for the operation because the tiles consist of LUTs, and different properties arise. The first one is that every operation takes the same amount of time. The second one is that if we store results on the same row, we can generate a total of 8 results at the same time. In the previous example, every adder is mapped on an individual tile, which results in a 3-input LUT—effectively using 16 bits or 25% of the total memory of a tile.
If we increase the operation to use the 4-input of the LUT effectively, we increase the total efficiency of the adder. For instance, we can calculate the result of adding 2 bits. The input of the LUT is then A_1, B_1, A_0, B_0 and the result is C_1, S_1, S_0. Table 5.4 shows the truth table of this 2-bit adder. The two-bit adder needs 48 bits or 75% of the memory of the tile. The adder can generate two results at the same rate it took the previous example to generate a single result.

Table 5.4: Truth Table of the 2-bit Adder

B1	A1	B0	A0	C1	S1	S0
0	0	0	0	0	0	0
0	0	0	1	0	0	1
0	0	1	0	0	0	1
0	0	1	1	0	1	0
0	1	0	0	0	1	0
0	1	0	1	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	0	0
1	0	0	0	0	1	0
1	0	0	1	0	1	1
1	0	1	0	0	1	1
1	0	1	1	1	0	0
1	1	0	0	1	0	0
1	1	0	1	1	0	1
1	1	1	0	1	0	1
1	1	1	1	1	1	0

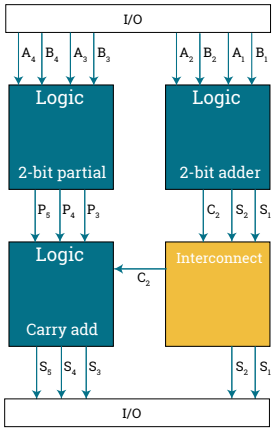


Figure 5.10: 4-bit adder constructed from 2-bit adders

To generate the result of the subsequent bits is less trivial because we have to account for the carry bit of the previous 2-bit operation if we want to generate the same three results but with a 5-input LUT. We need $3x2^5 = 94$ bits, which is larger than the 64 bits of the total memory of an 8x8 tile.This restriction is overcome by splitting the operation over two tiles, where the first solves a partial result and then accounts for the carry of the previous operation. Figure 5.10 shows the resulting arrangement of a 4-bit adder with 2-bit adders. We constructed two distinct adders with this architecture. The first closely

resembles a traditional ripple-carry adder. We mapped each adder onto individual tiles, akin to building a ripple-carry adder from traditional logic gates. Consequently, the total delay increases linearly with the width of the binary integers being summed. Additionally, we developed another design that utilizes the internal memory within each tile more effectively, which is anticipated to reduce the delay by minimizing the critical path influenced by the carry bit.

5.5.2. Load-store Architecture

In a load-store architecture, operations are carried out on data only when it is placed in specific storage areas called registers. This requires separate steps to move data from the memory to the registers, and then perform computations on it. In simple terms, this means that the data must be ”loaded” before any operations can be done, and the results must be ”stored” back in memory afterwards.
At its core, this architecture relies on several specific components working together. As depicted in Figure 5.11, the basic setup includes instruction memory linked to a register that holds a limited amount of data. This register is connected to a device called a shifter, which can move data bits to the left or right. Afterwards, the manipulated data is stored in a result register, from where it can be sent back as needed in the next clock cycle. Since this setup is quite simple, it doesn't include separate memory space for loading data; instead, it can directly use data stored in the instruction memory as immediate values.

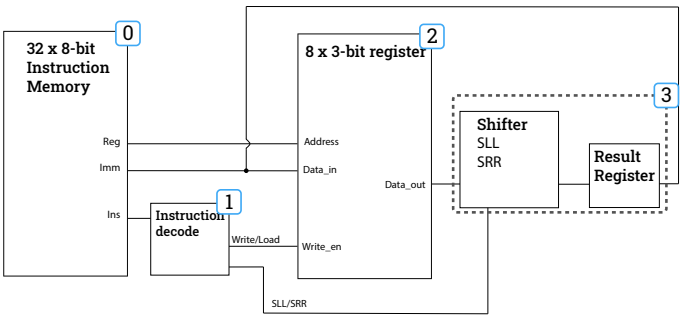


Figure 5.11: Basic Load-Store Architecture

The main objective of this benchmark study is to examine how the memory blocks function in this architecture, particularly focusing on the possibilities for enhancing and expanding the design using the fundamental components available. More specifically, we aim to demonstrate how connecting multiple memory blocks can potentially increase the total memory capacity, which is discussed in more detail in section 4.2.4.

Design In this benchmark, we are examining the architecture depicted in Figure 5.11. This design is made up of 4 main components, labeled from 0 to 3 within the blue boxes. The instruction memory (marked as ’0’) contains both the commands and the data that can be moved to the specified registers. This design allows for a variety of instructions, which are represented visually in Figure 5.12. We break down the 8-bit instruction into three parts: the first part contains the main instruction which is converted into control signals guiding the shifter and the register. Given that it is two bits, it allows for a total of four different instructions, as outlined in the table below:

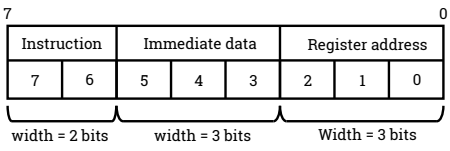


Figure 5.12: Representation of instructions for load store design

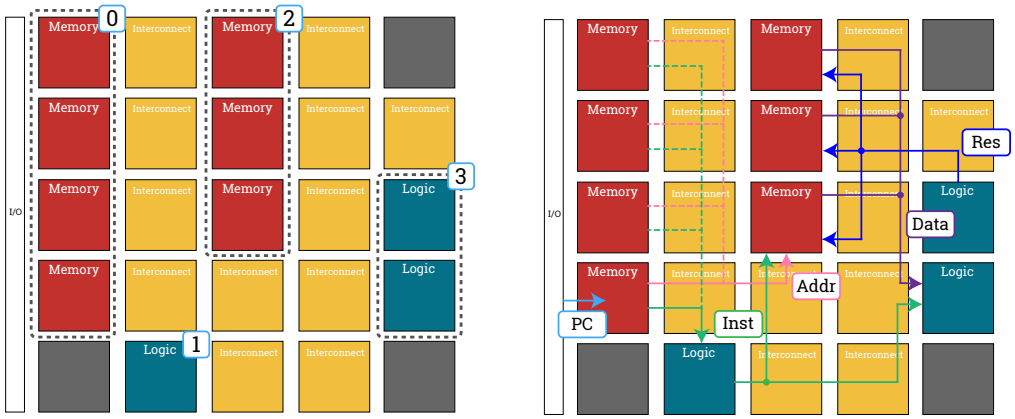
Table 5.5: Instruction Guide for Load-Store Design

In1	In0	Instruction
0	0	Shift logical left
0	1	Shift logical right
1	0	Load result in register
1	1	Load immediate value in register

It's important to mention that a store instruction was not included in this design due to the absence

of a separate data memory component. Despite this, the design successfully separates instruction sequences from memory processes, ensuring that calculations are only conducted on data housed in the registers.
The illustration in Figure 5.13 exhibits the design from Figure 5.11 transitioned into a unified tile structure. In Figure 5.13a, labels have been applied to highlight the separation of various components, which have been integrated as follows:

- **Instruction Memory (0):** This is comprised of four stacked tiles, each containing 8 rows of 8 bits. Therefore, the combined structure houses a total of 32 rows of 8-bit memory.
- **Instruction Decode (1):** This component splits the instruction into distinct control signals which are then forwarded to the register block and the shifter.
- **3-bit Register (2):** Here, we utilize three tiles set in memory mode, allowing for the simultaneous writing of 3-bits.
- **Shifter (3):** This is a logic block capable of conducting a logical shift operation on the 3-bit data received from the registers.



(a) Design mapped on unified tiles

(b) Representation of the data/control flow

Figure 5.13: Simple load-store on unified tile

In Figure 5.13b, we illustrate the data and control flow within the current design. The process initiates with a signal from the IO - referred to here as the Program Counter (PC). This signal infiltrates the lower memory blocks before progressing upwards to the successive blocks layered above. The PC value essentially dictates the instruction selection in the memory address. To elaborate, a PC value of 20 would indicate the selection of the 20th row starting from the base layer. Given the stacked configuration of the memory blocks, this translates to the selection of the fourth row in the third block from the bottom. Subsequently, this data is channeled to the interconnected blocks situated to the right of the memory blocks.
An interesting aspect to note here is the logical OR operation performed as the signals traverse through the interconnect block. This prevents the need for separating the signals emanating from each individual memory block. The instruction procured from the memory undergoes a split into two sections: the core instruction part and a segment used to control the register address and immediate value. The instruction segment, denoted as 'Inst', navigates towards a block set in logic mode, which then births control signals destined for the shifter and the registers.
Following this, the address signal spearheads towards the register blocks, instigating the release of a specific bit present in the designated row. This data journeys to the shifter where it undergoes transformation, culminating in a result that is conserved within the flip-flops situated at the output of the LUT. Depending on the initial instruction, this data is then aligned on the pathways leading to the registers.

Concurrently write multiple bits in the memory blocks One limitation of this architecture is that a memory block can only write a single bit at a time to the crossbar array, which can be a bottleneck when there's a need to write multiple bits simultaneously. Nevertheless, there are several strategies to overcome this restriction within the existing framework. One effective solution, as showcased in this design, is to use several tiles configured in memory mode. This approach involves vertically distributing the data across multiple tiles, enabling parallel writing to the crossbar arrays by leveraging the individual writing mechanisms of each tile.

5.6. Conclusion

In this chapter, we presented the method of verification and the methods used to determine the system characteristics. We utilized an event-based simulator based on the SystemC framework. This simulator aims to simulate each tile's output based on the response of neighbouring tiles or input. The simulator also implements a timing model, which enables the simulator to give a cycle-accurate prediction of the system's behaviour.

At the core of our simulator are two models of the Unified Tile. The first model is a behavioural model which can generate expected outputs of a tile based on the state of the tile, the content of the memory array and the provided input. The second model is an analytical model of the electrical implementation of the different components of the tile. The analytical model can predict a tile's static and dynamic energy usage.

Delving deeper into the first model, the model consists of multiple smaller functions that can implement a tile's main behavioural characteristics. The behavioural model implements the three operational modes and can generate the expected outputs. The behaviour depends on the content of the memory array, so one of the main contributing factors of a tile is the content of this array, which is thus also one of the parameters which can be given to the system description readable by the simulator. Because the simulator simulates the interaction of multiple tiles, which also considers timing, a cycle-accurate representation of the system behaviour is achieved.

The analytical model estimates an individual tile's static and dynamic energy usage. This model is based on parameters generated via SPICE simulations of the presented circuits from Section 4.3 and characteristics obtained from literature. The static energy can be determined by extrapolating these parameters to the size requirements of an application, considering the desired size of a tile and the number of tiles needed for an application.

Dynamic energy is extrapolated from the size requirements and the input stimuli associated with an application. Therefore, there is a connection between the behavioural and analytical models. The behavioural model accurately represents the interactions of each tile with IO and neighbouring tiles. Based on our understanding of the switching behaviour associated with the used components and the provided input signals to each tile, an accurate estimation can be made about the dynamics of a tile.

The behavioural and analytical models comprehensively understand tiles' key metrics like energy usage, latency and area utilization. These key metrics are used as a solid foundation for functional verification and analysis of the system.

The generated results are then used to explore how different eNVM technologies perform to understand the design space better. A comparative analysis, anchored in this comprehensive methodology, was discussed. This analysis serves dual purposes: understanding the best-suited memory technology for the unified tile architecture and benchmarking our design against state-of-the-art architectures with emerging eNVMs. The chapter also highlights the employment of micro-benchmarks to evaluate the performance across different memory technologies.

To compare this architecture with other eNVM-based architectures, we used a comparative analysis framework for eNVM-based systems. This comparative framework allows us to compare the different architectures based on the needed components associated with eNVM-based systems.

We created two benchmarks which are used to generate relevant metrics. The first benchmark is an Adder used to compare the different memory technologies and the state-of-the-art architectures. We compare results from literature for the other architectures with results from our simulator. The second benchmark, a Load-store architecture, uses the full range of the tile's functionalities and is not an example grounded in logic. This benchmark is used to compare the different memory technologies with each other but not to compare to other state-of-the-art architectures.

In closing, this section detailed our methodical approach to system verification using an event-based

simulator rooted in the SystemC framework. Our dual-model strategy, consisting of a behavioural model for tile interactions and an analytical model for energy estimation, provides a detailed understanding of the Unified Tile. Introducing two benchmarks—the Adder and Load-store architecture—enriches our analysis by comprehensively understanding the system's capabilities and performance. Drawing from these models and benchmarks, we have also presented a comparative study with other eNVM-based architectures, showing our design's strengths and potential areas of improvement.

6
Results

This chapter presents the results obtained from the simulator-based architectural and circuit designs presented in Sections 4.2 and 4.3. The simulator parameters are derived from existing literature and SPICE circuit simulations. These parameters are presented in Section 6.1, and form the basis for analysing the system's electrical performance. Based on those parameters, we can extrapolate the dynamic energy consumption of the benchmarks presented in Section 5.5. Furthermore, a brief presentation on the verification of the simulator's outputs is provided in Section 6.2. The chapter concludes with a comprehensive comparison of the various memory technologies in Section 6.3, and a comparison with the current state-of-the-art architecture in Section 6.4

6.1. Simulator parameters

In Section 5.3, we described two primary operations performed on the crossbar array and embedded memory devices: sensing and programming. This section discusses the power consumption data acquired from SPICE simulations and literature for the circuits presented in Section 4.3. These results are used to extrapolate the results for the energy efficiency, presented in Section 6.3.

Sensing The structure of each tile is essentially an NxM grid formed by an array of FeFET devices. Here, 'N' denotes the number of rows and 'M' denotes the number of columns. A sensing amplifier is connected to each column to sense the state of the FeFET device. In this analysis, we focus on the energy characteristics of a single memory element, specifically while sensing a single FeFET in a column. Figure 4.18a in Section 4.3 depicts the related circuit diagram.
During our simulation, set with a VDD of $3.3\,\mathrm{V}$ and a feature size of $90\,\mathrm{nm}$, we noted the following characteristics in terms of power consumption:

- **Static Energy Usage:** The static power consumption is $84.64\,\mathrm{pW}$.
- **Dynamic Energy Usage:** The energy consumption during reading operation is as follows:
  - Reading a '0' $2.21\,\mathrm{fJ}$.
  - Reading a '1' $5.11\,\mathrm{fJ}$.
- **Operational Speed:** The sensing process exhibits an delay of $81.9\,\mathrm{ps}$.

These results are comparable with the paper's results where the sense amplifier's design is based [17]. We performed a similar analysis for the other memory technologies. Table 6.1 gives an overview of the results of this analysis.

The results from this table are for a single column N. A tile consists of multiple columns connected with each a sense amplifier. The simulator considers the operation results based on the total energy usage of columns activated; thus, when multiple bits are read from multiple columns, the energy usage of a single tile is the sum of all these read operations. We consider the selection circuit for the WL and BLs as a set value regardless of the chosen technology. Switching the selection circuits takes for an 8-bit selection circuit 8.82fJ with a delay of $14.14\,\mathrm{ps}$.

Table 6.1: Electrical Characteristics for an 8-bit column

Metric	**FeFET**	**ReRAM**	**MTJ**	**SRAM**[1]
Reading '0'	2.21 fJ	4.78 fJ	5.82 fJ	6.42 fJ
Reading '1'	5.11 fJ	10.5 fJ	9.16 fJ	110 fJ
Reading delay	82 ps	162 ps	116 ps	306 ps

[1] SRAM uses a differential amplifier instead of the amplifier presented in 4.3.3

Programming Many factors can influence the time needed to program these emerging eNVM devices. Think of the size of the transistors or the used doping. For our comparison, we concentrated on the structure given in [17], which used the results from [6] and [9] for ReRAM and FeFET, respectively. We extended these results with data from table 3.1 for the STT-MTJ technology. We obtained the following metrics we used for the simulator.

Table 6.2: Electrical characteristics programming 1-bit

Metric	**FeFET**	**ReRAM**	**MTJ**	**SRAM**
Delay	1 μs	10 ns	5 ns	27 ps
Energy	53.88 fJ	625 fJ	100 fJ	1.54 fJ

Conclusion for Simulator Parameters The parameters for the simulator were determined from a combination of literature references and SPICE simulations. The metrics focused on the crossbar array's sensing and programming (read/write) operations and embedded memory devices. We also run SPICE simulations for the peripheral circuitry, which completes the needed variables for the method presented in 5.4.1. The data reveals the differences in energy consumption and operational delays for various memory technologies. In the context of sensing, the results underscore the FeFET device's efficacy compared to the other technologies, evidenced by their relatively lower energy consumption and shorter sensing delay times.
Similarly, in the programming of these technologies, there exists a noticeable variation in delay and energy consumption among the memory technologies. From the parameters for programming the devices, we can determine that FeFET is not up to par with the other memory technologies.

6.2. Verification

The following subsections delve into the verification processes for the two micro-benchmarks: the ripple-carry adder and the load-store architecture. For both components, directed testing was used —a method wherein specific test scenarios are executed based on the system's expected behaviour. The designs' outputs were observed and compared to the expected results through these tests. The equivalency between the expected and observed outputs determines the functional correctness of the designs.
Furthermore, a verified functional behaviour paves the way for energy consumption evaluations in subsequent sections. Based on a verified design, any conclusions from power efficiency, latency, or other performance metrics are more reliable. Hence, the role of verification is not just to establish functional correctness but also to lay a foundation for subsequent analyses in Section 6.3.

6.2.1. Adder

This section discusses the results from the ripple-carry adder presented in Section 5.5.1. The presented adder is 4-bit, meaning we can add two 4-bit integers. Table 6.3 shows some results obtained from simulation as presented in Section 5.2.

Table 6.3: Test sequence simulated at 10kHz

Step	A	A (BIN)	B	B (BIN)	Expected	Expected (BIN)	Obs. (BIN)
1	0	0000	0	0000	0	00000	00000
2	1	0001	0	0000	1	00001	00001
3	0	0000	1	0001	1	00001	00001
4	1	0001	1	0001	2	00010	00010
5	3	0011	3	0011	6	00110	00110
6	15	1111	0	0000	15	01111	01111
7	0	0000	15	1111	15	01111	01111
8	15	1111	15	1111	30	11110	11110
9	15	1111	14	1110	29	11101	11101
10	12	1100	12	1100	24	11000	11000
11	0	0000	0	0000	0	00000	00000

The results obtained from the simulation show the expected output for the provided input stimuli.

6.2.2. Load-store Architecture

Section 3.2.1 discusses the load-store architecture implementation. In that section, we explained that the instructions which control the different parts are saved in a stack of 4 memory blocks. The program counter sequentially selects the row of the instruction memory. We use a sequence of instructions to verify correct behaviour and manually check the signals going out of the logic block.

Table 6.4: Test sequence simulated at 10kHz

Step	Instruction	Ins. (BIN)	Expected (BIN)	Obs. (BIN)
1	LOAD 010 R1	1101000	000	000
2	LOAD 111 R2	11111010	000	000
3	LOAD 011 R3	11011011	000	000
4	SLL R1	00000001	000	000
5	LOAD RES R1	10000001	001	001
6	SLR R2	01000010	000	000
7	LOAD RES R3	10000011	110	110
8	SLL R3	00000011	000	000
9	LOAD RES R3	10000011	011	011

Table 6.4 shows a used test sequence to test the functional correctness of the architecture. In the column expected, the binary values are given of the expected output of the shifter. Note that the signals are always a clock cycle delayed. The signals behave as expected, which shows that we can also simulate the behaviour of memory and interconnect blocks.
The verification process affirms that our simulator can produce accurate and verifiable results. Moreover, the behavioural model of the unified tile can determine the correct system behaviour. This establishes a foundation for subsequent electrical and functional evaluations and analyses.

6.3. Efficiency Analysis

This section presents the benchmark results using the unified tile model and the simulator. As previously discussed, the simulator has proven its ability to generate valid results based on the unified tile's behavioural model. This model can determine the tile's dynamic switching behaviours without executing a circuit-level simulation of the crossbar array in a SPICE simulation.
In this analysis, we will focus on the electrical characteristics of the architecture. We will emphasise the FeFET memory device, which we will compare to other eNVM and SRAM devices to evaluate its performance.

6.3.1. Ripple-Carry Adder

The Ripple-Carry Adder utilises four tiles, each producing two outputs simultaneously. The test from table 6.3 shows an average dynamic power consumption of 1.115 pJ/Ops. The dynamic energy usage ranges from a minimum of 1.024 pJ/Ops. to a maximum of 1.326 pJ/Ops.
Figures 6.1 and 6.2 show the total energy consumption pattern for the ripple-carry adders of varying sizes and frequencies. Note that the logarithmic scale on the left axis and the omission of specific configurations which are infeasible at higher frequencies.
In Figure 6.1, energy consumption per operation stabilises at higher frequencies. In contrast, Figure 6.2 shows an uptick in total power consumption beyond this stabilisation, attributed to the dynamic energy significantly surpassing the static energy consumption.

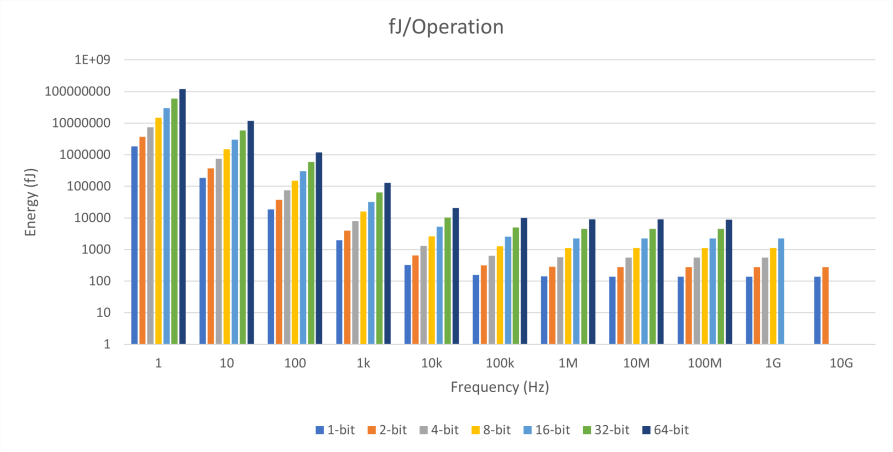


Figure 6.1: Worst-case Energy per operation

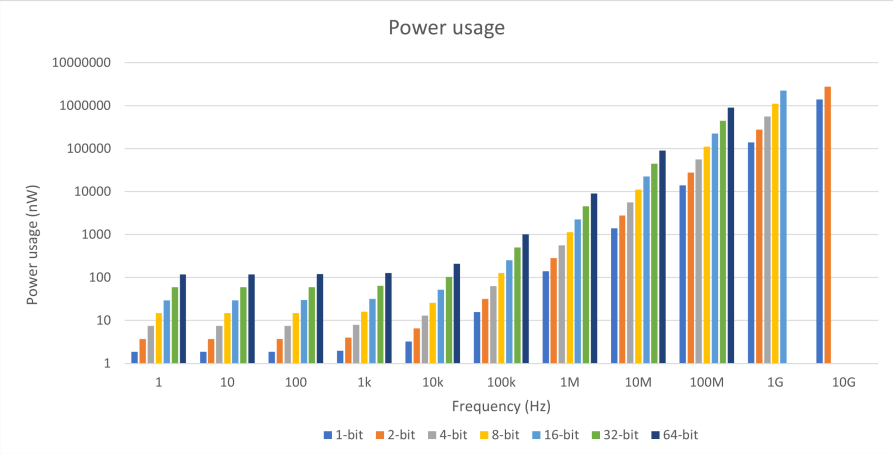
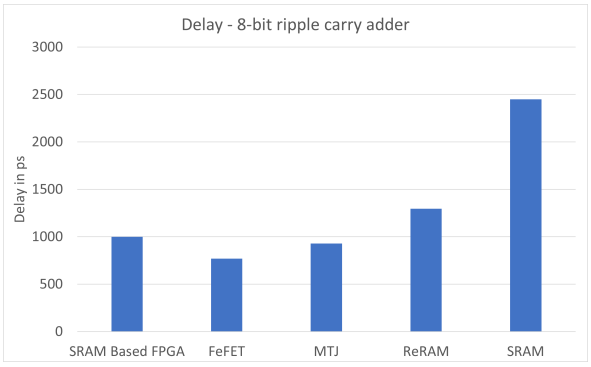
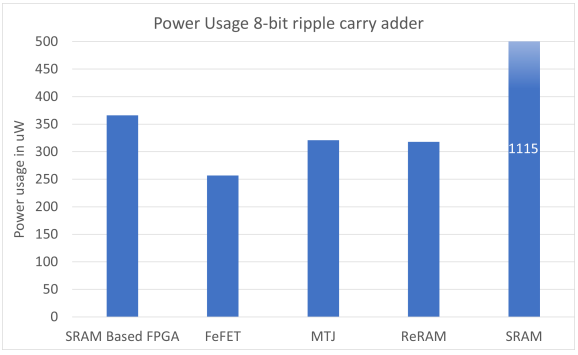


Figure 6.2: Worst-case energy usage

Memory Technology Differences The use of different memory technologies will impact delay and power consumption. This section will compare various technologies, referencing results from an SRAM-based FPGA 8-bit adder implementation results obtained from [23]. Figure 6.3 showcases delay and power usage for the 8-bit adder benchmark for different eNVM technologies simulated at 1 GHz. Fe-FET has the best delay and power performance, followed by STT-MTJ and ReRAM technology. When applied to this architecture, SRAM significantly underperforms its SRAM-based FPGA counterpart.



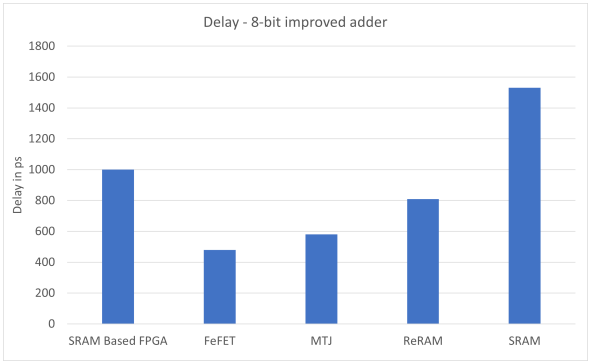
(a) Delay comparison of 8-bit adder



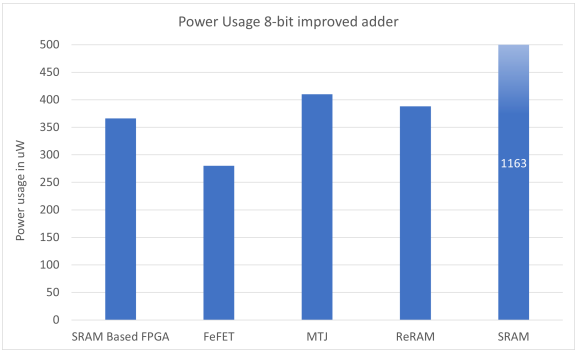
(b) Power comparison of 8-bit adder

Figure 6.3: Differences between memory technologies 8-bit adder

Improved Adder An improved adder design, previously discussed in Section 5.5.1, utilises resources more efficiently by combining multiple operations in a single tile. Effectively, this results in a shorter critical path. Figure 6.4a shows the results of this improved version. The figures show that the improved adder improves the execution time of all technologies. This makes this design outperform the FPGA with the three emerging eNVM technologies. Note that the energy consumption goes up for all technologies with this configuration. This uptick is caused by the increase in read operations from the memory. In Section 6.4, a more detailed analysis of the computational steps related to the dynamic energy usage.



(a) Delay comparison of 8-bit improved adder



(b) Power comparison of 8-bit improved adder

Figure 6.4: Differences between memory technologies 8-bit adder

Conclusion This section explored the delay and power usage for an 8-bit adder design, identifying a significant surge in power consumption when dynamic power eclipses static power. The performance of two adder designs—a basic 8-bit ripple-carry adder and an improved version—was examined across various memory technologies, using an FPGA as a baseline. The FeFET and STT-MTJ technologies outperformed the FPGA in both delay and power. We showed the results of two benchmarks one less optimized ripple-carry adder and an optimized adder. The difference between the two is significant, which is expected. The difference highlights the relation between how an application is mapped and the observed performance. As noted earlier, the improved adder has a slight increase in average power usage caused by increased read operations. Figure 6.5 displays the normalised power-delay products

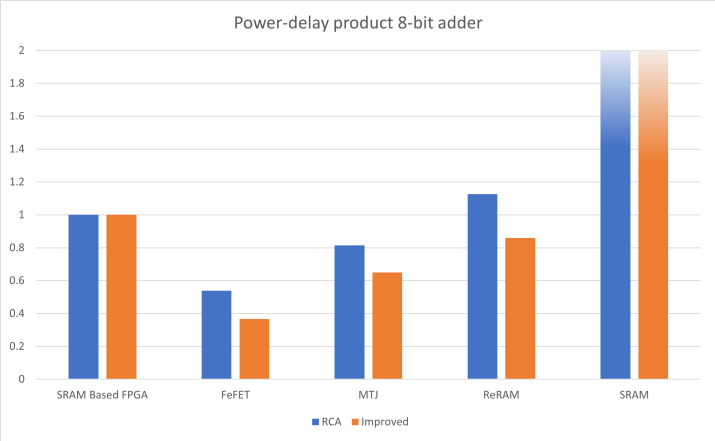


Figure 6.5: Comparison of power-delay products 8-bit adder

for all the examined implementations. The PDP shows that the power increase to reduce delay is not out of proportion. This underscores the advantages of improving the application in both speed and energy efficiency.

6.3.2. Load-store Architecture

This section delves into the Load-store benchmark. Unlike the previous benchmark, which consisted exclusively of logic-configured tiles, this benchmark uses memory reads/writes, routing, and some logic blocks. This configuration leads to noticeable differences in power consumption due to the increased delays and energy demands for programming individual cells.
The writing speed of the technology is the main restricting factor when it comes to execution time. These writing speeds differ vastly between technologies. From our analysis from Section 6.1, it is evident that the writing speeds of FeFET devices are significantly lower than the other technologies, where SRAM still offers the fastest switching speeds. Comparing the different technologies at the same frequency is not fair. Because FeFET will offer better energy performance at lower clock frequencies than SRAM, but SRAM can be used at significantly higher operating speeds. To better study the differences, we present the maximum operating frequency this micro-benchmark can operate based on the used memory technology. These operating frequencies are displayed in Table 6.5.

Table 6.5: Max operating frequency different technologies load-store architecture

Technology	Delay LCP	Max Frequency
FeFET	1 μs	1 MHz
ReRAM	11.1 ns	90 MHz
MTJ	5.8 ns	172 MHz
SRAM	3.7 ns	272 MHz

With the simulator we obtained the results of the four technologies at different operating frequencies. These results are given in the plot in Figure 6.6, note that the x and y axis are both in log scale. Some technologies do not pass verification at higher frequencies at are therefore omitted from the graph.
The results show that SRAM is capable of reaching higher frequencies but that at low frequencies the technology operates at much higher power usage. Which is as already discussed with the previous benchmark caused by the much higher static energy usage of SRAM cells. For the emerging eNVM technologies it is apparent that at these frequencies the dynamic energy usages is the main contributing factor to overall systems power usage.
This benchmark highlights that the selected technology and the use of writes heavily influence the maximum operating speed of the implementation. In Figure 6.7a, the best-case energy usage per operation is displayed. Based on these best-case metrics, we can determine the energy-delay product of the different technologies, depicted in Figure 6.7b. The EDP clearly shows the relation between the lower

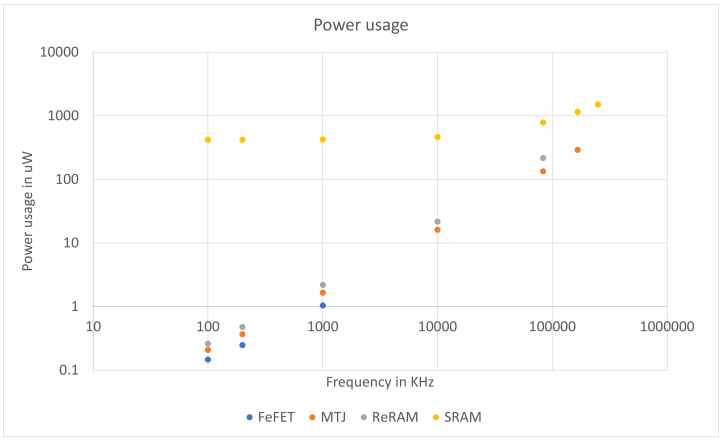
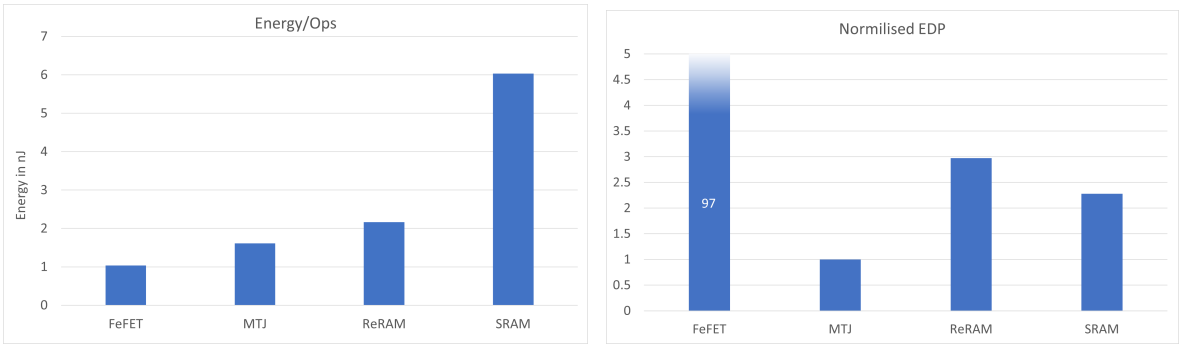


Figure 6.6: Power usage at different operating speeds Load Store benchmark

operating speed and dissipated power. This metric shows that STT-MTJ offers the best trade-off of the emerging eNVMs in maximum operating speed and power usage. If we want to optimise purely on energy usage, FeFET still offers the best performance, which will impact the delay of the application.



(a) Best-case energy usage per operation load-store architecture

(b) Energy-delay product load-store architecture

Figure 6.7: Differences between memory technologies load-store architecture

6.3.3. Conclusion

This section discussed the energy efficiency results based on the micro-benchmarks and the unified tile model. These results gave us insights into the performance of the different memory technologies when used in this architecture. The first showed the differences between the technologies when configured in a logic-only application. The second benchmark shows the performance in a more memory-heavy application.

In the case of the adder benchmark, we showed that FeFET and STT-MTJ technologies demonstrated a notable performance advantage over the FPGA, both in terms of speed and power metrics. For the Load-store architecture benchmark, several key observations were made. Primarily, the write speeds of the chosen memory technology emerged as the determining factor for the system's maximum operating frequency. In its capacity to reach significantly higher frequencies, SRAM offers the lowest delays. However, SRAM also suffered from increased power usage due to its higher static energy consumption. For the emerging eNVM technologies, dynamic energy consistently contributes to the overarching power consumption. STT-MTJ distinguished itself as the most balanced memory technology because of its balance between operational speed and power consumption when read and write operations are considered. In contrast, while FeFET's writing speeds lagged, its supremacy in terms of energy performance remained.

6.4. Comparison state-of-the-art

To compare this architecture with the other state-of-the-art ReRAM-based architectures Liquid Silicon and Merging Everything, we use the comparison framework presented in Section 5.4.1. Comparing these architectures is not trivial because they fundamentally differ from one another. The architecture as presented in this thesis, for instance, employs the same high-level concept of merging the three key functionalities of an FPGA in one unified tile like Merging Everything and to some degree of Liquid Silicon, but the internal mechanics to do so are less comparable.

6.4.1. Computational Steps

Merging Everything and this architecture are more alike because information is processed via LUTs. Processing this way will reduce the number of sequential processing steps needed to come to a result because everything is pre-calculated. The downside is that the added hardware to control the crossbar array causes longer delays.

For our comparison, we use the adder micro-benchmark. In Section 5.5, we explained this benchmark's structure, which consists of utilising four tiles, each capable of adding 3 bits. Leading to a value for N_{cs} of 4 for each addition of 3 bits needs two outputs (the sum and the carry). This results in a N_{read} of 2. There is no write to the values in the memory grid; thus $N_{write} = 0$.

For Merging Everything, determining these values is also straightforward; the main difference with our architecture is that in the Merging Everything architecture, multiple functions reside in a single tile. This difference on the higher level does not change the number of processing steps because we still need 4 sequential processing steps in this architecture; thus $N_{cs} = 4$. Each processing step generates two values, the carry bit and the sum result, thus $N_{read} = 2$.

Liquid Silicon differs as an architecture because it processes information based on logic-in-memory principles. This comes down to primarily processing signals by AND, NAND, OR and NOR operations. The number of processing steps increases because when restricted to the logic operations mentioned above, for 1-bit full adder 6 sequential steps with a total of 9 reads are needed to complete the single operation, which gives us a $N_{cs} = 24$ and $N_{read} = 9$. In table 6.6, we provided an overview of the values of an 8-bit addition. We also included the values for the improved adder, which is also discussed in Section 5.5

Table 6.6: Overview of processing step values for each architecture for 8-bit adder example

Architecture	N_{cs}	N_{read}	$N_{read}(N_{cs})$	Critical Path
This work - ripple carry adder	8	2	16	8
This work - improved adder	8	3	24	5
Merging Everything [1]	8	2	16	?
Liquid Silicon [2]	48	1.5	72	?

6.4.2. Delay

Processing steps allow us to evaluate the complexity of implementing functions independent of the implemented circuits. However, the implemented circuits still impose differences across the architectures. Thus, we still consider the real delay metrics for a fair performance comparison. Liquid silicon can execute a computational step much faster because it needs less control and peripheral circuitry. The average delay of an operation for each architecture is given in Figure 6.8. This comparison includes the results from [23]. This graph shows that this architecture has the same delay as the other architectures.

6.4.3. Power efficiency

Energy consumption for different sizes RCA is given in Section 6.3.1 but to compare the different architectures with each other we need to compare via the framework presented in Section 5.4.1. Analytically we E_{sa} differs based on the previous state of the SA and the state of FeFET cell as observed in Section 6.1. In Figure 6.9a are the power results obtained from [23] for ME and liquid silicon and simulation for the results of this work. Note the differences in power usage for the improved adder and the simple ripple carry adder. The higher power usage is an result of the increase in read operations (N_{read})

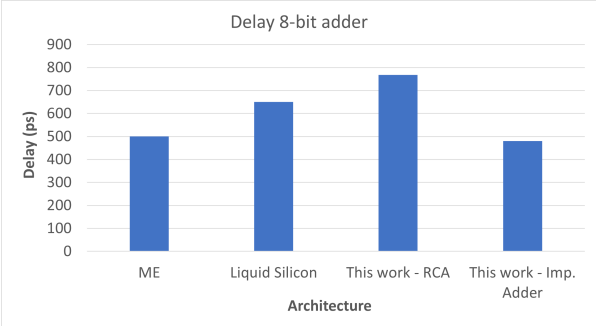
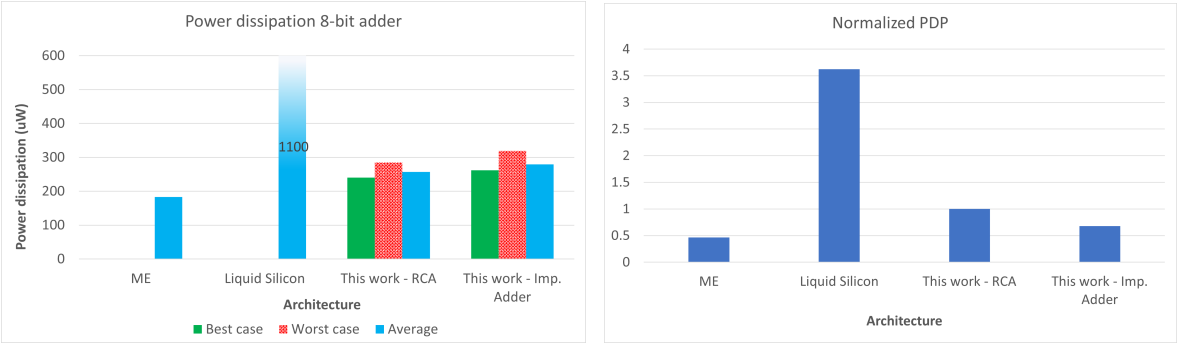


Figure 6.8: Delay comparison for 8-bit adder

needed to complete the computation. In figure 6.9b we show the normalized power-delay product of these architectures.



(a) Power comparison of 8-bit adder

(b) Normalized Power-delay product comparison of 8-bit adder

Figure 6.9: Electrical characteristics 8-bit adder

6.4.4. Area

To compare area, we take into consideration three aspects: the number of memory cells A_{cells}, the number of cells needed for the periphery A_P and for the controlling circuitry A_C. The area of the memory cells differs for the FeFET-based and ReRAM-based systems, so we define two cell definitions called $A_{CellsReRAM}$ and $A_{CellsFeFET}$. The comparison tiles are used 16x16 for Merging Everything and Liquid Silicon and 8x8 for the example of this work, which gives a value of 256 for Merging Everything and Liquid Silicon and 64.

Because the architectures compared in this work rely on routing between tiles over the crossbar array or pre-programmed routes over ReRAM/FeFET-based muxes, the need for controlling circuitry is almost zero. We do not consider those structures in this work except for the programming routines needed to program the different memory technologies. So, we assume $A_C = 0$ for the three architectures we compare. Because all architectures need programming circuitry, we can still fairly compare the algorithms by only considering A_{cells} and A_P.

To determine A_P, we estimate the total amount of cells used based on the total number of cells used for internal routing, sensing circuits and the logic used at the input and outputs of the memory array. We make a distinction for each architecture based on the used technology, which gives us the following definition of A_P:

$$A_P = A_P FeFET + A_P ReRAM + A_P CMOS \tag{6.1}$$

$A_P FeFET$ represent FeFET devices, $A_P ReRAM$ represents ReRAM devices, and $A_P CMOS$ is the total number of PMOS and NMOS devices used in the periphery circuit. In this comparison, we estimate the number of transistors needed in MUXes, a d in the case of the FeFET or ReRAM muxes, the auxiliary transistors needed to program individual cells.

This Work An 8x8 tile used in this thesis, the periphery consists of 8 sense amplifiers and a vertical selection based on AND gates and horizontal selection based on AND gates and transmission gates to accommodate a programming structure. These selection circuits have 8 lines for the 8 word- and bit-lines. Furthermore, do we consider the MUXes needed to route signals to one of the four directions?

Table 6.7: Device count for this work

	CMOS	FeFET
A_{cells}	64	64
A_P	626	112
Total	690	176

Merging Everything The 8-bit adder benchmark used a single tile 16x16, so we used this size to evaluate the tile size. The architecture of Merging Everything is different from this work in that it relies more heavily on internal routing. Like our design, every column has a sense amplifier in Merging Everything. However, Where Merging Everything differs from this work is that a result can be internally routed to other LUTMUXes from other columns of MUX. This makes it possible to execute 16 concurrent functions inside of a tile. It is also possible to sequentially order LUTs inside of the tile to each other to create more complex functions. Multiple tiles achieve the same for the designs presented in this thesis and liquid silicon. The tile size fits the application perfectly, as we need 16 functions to execute an 8-bit ripple-carry add operation.
Because the researchers did not provide a circuit of how they implemented the ReRAM MUXes, we assume the internal routing is implemented as a crossbar array, which allows a more efficient programming structure and more area-efficient placement of the ReRAM devices. In total, we get the following values for merging everything architecture.

Table 6.8: Device count for Merging Everything

	CMOS	ReRAM
A_{cells}	256	256
A_P	5104	2224
Total	5306	2480

Liquid Silicon Liquid silicon relies less on peripheral circuitry but uses more hardware to connect the tiles' nodes. An overview of the used hardware structures is given in [20] and [22]. The reduction in peripheral circuitry is because there is less hardware for the selection and internal or external routing. Data flows solely over the memory array. The following results for the area are estimated:

Table 6.9: Device count for Liquid Silicon

	CMOS	ReRAM
A_{cells}	0	256
A_P	1984	320
Total	1984	576

Concluding remarks Area For a fair comparison between the three architectures, we have to adjust for the grid size and the total tiles needed to accomplish the same function. We know from Section 5.5 that eight tiles are needed for an 8x8 tile from this work. We assume that a single 16x16 tile is needed in Merging Everything because we need 16 functions to generate the sum result and the carry bit. Liquid Silicon is less trivial because we must map the 48 computational steps. It is impossible to do this on a

single tile because we can not route inside of a tile. We need multiple liquid silicon tiles to do operations in sequence. In the best-case scenario, we can map the adder on just 4 tiles. This gives us the adjusted table for different architectures:

Table 6.10: Comparison different architectures

	This work	Merging Everything	Liquid Silicon
CMOS	5520	5292	7936
FeFET	1280	0	0
ReRAM	0	2480	2304
Total	6800	7772	10240
	18320 F	15312 F	17152 F

This architecture reduces the total device count compared to the other architectures, which is expressed in a slight improvement over merging Everything of 1.14x and 1.5x over liquid silicon. Adjusted for the device size obtained from the table 3.1 from Section 3.1, this work has a higher area utilisation than the other architectures.

6.5. Conclusion

In this chapter, we evaluated our proposed architecture's electrical performance area and tested the functional correctness of the benchmarks. We discussed the distinct benefits of the FeFET technology, notably its higher energy efficiency than other eNVM technologies when used for read operations. However, we must consider the significant programming latency associated with this memory technology. This latency is particularly detrimental in applications with frequent write operations. Contrarily, the STT-MTJ technology showcased significantly better write speeds, positioning STT-MTJ as a more balanced option of the three emerging eNVM we discussed.

Our assessment included a comparison of the state-of-the-art architectures, such as Merging Everything and Liquid Silicon, and a comparison with an FPGA based on SRAM technology. Our architecture closely mirrored Merging Everything in terms of delay but fell short in improving power dissipation. Data from the literature and our findings show that Liquid Silicon is significantly underperforming in delay and power consumption. Against an SRAM-based FPGA, our model exhibited a clear advantage in speed and delay metrics.

Applying a LUT-based approach to implement configurable logic has a clear advantage. The LUT-based approach can reduce the complexities of the amount of sequential steps taken, which results in a lower number of read operations. Dynamic energy usage is coupled with the amount of read operations. Even though LUTs necessitate additional peripheral circuitry, this approach improves the total energy dissipation of the architecture. Comparing Merging Everything and our architecture against Liquid Silicon, A difference of a factor ranging from 3.5X to 7X PDP improvement is observed.

Based on the 8-bit adder micro-benchmark, our architecture boasted the lowest device count, with Liquid Silicon being the most resource-intensive. However, accounting for size discrepancies between ReRAM and FeFET, our design commanded a higher area footprint.

Based on our analysis, we have found that developing an RCA by utilizing eNVM technologies has better performance than SRAM-based FPGAs. Our proposed architecture minimises the device count while maintaining delays similar to the current state-of-the-art Merging Everything.

7

Conclusion

The main goal of this thesis was addressing the following research question: How can we create a reconfigurable hardware structure utilizing emerging eNVM technologies that is capable of flexibly and interchangeably operating as functional, routing, and memory structures? We delved into current state-of-the-art architectures, subsequently proposing our own model accompanied by a robust simulation-based evaluation methodology. This concluding chapter further discusses the primary research question, emphasizing the challenges encountered and potential avenues for future exploration in this domain.

7.1. Summary

FPGAs are the most common example of fine-grain reconfigurable computing architectures. FPGAs allow hardware designers to (re)configure the datapaths implemented on the chip after production. This fundamentally differs from conventional computing systems like PCs or mobile phones, where programs are composed of sequentially performed instructions. The advantage of FPGAs is that changing the ”hardware” for the needs of an application can significantly increase the efficiency of that application.
Common FPGAs utilize an island-style topology consisting of CLBs and memory components arranged in a fixed pattern. A predominant feature of these advanced, fine-grain RCA platforms is their reliance on SRAM cells for storing configurations of both logic elements and interconnects. While SRAM technology provides swift access times, it inherently suffers from non-volatility, leading to continuous current leakage and associated power consumption. Simultaneously, the predefined structure governing the relationship between memory and CLBs introduces inherent constraints on resource availability. This rigidity can inadvertently introduce longer signal propagation delays as signals must traverse extended paths across the chip, potentially compromising overall system performance.
eNVM technologies are positioning themselves as potential alternatives to conventional SRAM in computing systems, particularly emphasising reducing power consumption. Our research shows that these eNVM technologies are promising. STT-MTJ has the most superior endurance in comparison with the other two technologies. ReRAM, with its distinct two-terminal structure, offers unique advantages in some designs, primarily when used in routing. At the same time, FeFET, despite its endurance challenges, showcases merits in both power and speed. Our investigations suggest that as these eNVM technologies continue to evolve and refine their characteristics, they stand poised to offer robust and efficient replacements for SRAM in the near future.
Two fine-grain RCA architectures are rising from these emerging eNVM advancements: Liquid Silicon and Merging Everything. Both architectures harness a 2D homogeneous grid topology, but their implementation of the core functionalities of the RCAs differ. Liquid Silicon uses its crossbar array and implements the different requirements via its connection nodes, which is a straightforward method without other components in the periphery of the crossbar array. However, concerns arise regarding its scalability and potential performance limits. Conversely, Merging Everything leans heavily on a LUT-based design. Using the same kind of crossbar array but with the added peripheral circuitry to implement LUTs based on the data in the crossbar array. With a pronounced focus on LUTs and internal routing. Merging Everything approach has been proven to be better equipped for implementing complex structures and

functions.
Building upon the FeFET eNVM technology, we have introduced a distinctive RCA architecture that employs a 2D homogeneous layout. Drawing inspirations from the Liquid Silicon and Merging Everything architectures. Central to this is our 'Unified Tile', which seamlessly integrates the three fundamental functionalities of a fine-grain RCA: memory, interconnect, and logic. The tile's operational modes are determined by its memory content and input routing, enabling flexible reconfiguration of the system's resources. Our design goes beyond a mere replacement of SRAM of the components of an FPGA, highlighting the potential and promise of eNVM technologies in future RCA designs.
We discussed our system verification approach, utilizing an event-based simulator developed within the SystemC framework. The simulator is built around two core models of the Unified Tile. The behavioural model determines the expected outputs of a tile-based on its state, memory array content, and input. This works with the analytical model that predicts the tile's static and dynamic energy consumption, drawing parameters from SPICE simulations and related literature.
We introduced two benchmarks: the Adder and the Load-store architecture. These benchmarks are essential for evaluating the performance of different eNVM technologies within our architecture. By comparing results from existing literature with our simulator's outputs, they allow a direct comparison with current state-of-the-art architectures.
This methodology provides a comprehensive insight into the Unified Tile's capabilities and performance. The combination of detailed models, strategic benchmarks, and comparative analysis underscores the strength of our design and identifies potential areas for future research.
We systematically evaluated our proposed architecture's electrical performance, functional accuracy, and area metrics against emerging eNVM technologies. The FeFET technology displayed superior energy efficiency in sensing operations but exhibited substantial programming latency, presenting challenges in applications with frequent write demands. In contrast, STT-MTJ demonstrated optimal write speeds among the eNVM technologies examined.
Our analysis also compared architectures, specifically Merging Everything, Liquid Silicon, and an SRAM-based FPGA. Regarding delay metrics, our architecture aligned closely with Merging Everything but did not achieve improved power efficiency. Using the adder micro-benchmark, our design showed a lower device count, although the more considerable size variance between ReRAM and FeFET resulted in a more significant area footprint. In computational efficiency, our architecture and Merging Everything outperformed Liquid Silicon, which is best reflected in the decreased PDP. In conclusion, our results indicate the feasibility of developing an RCA based on eNVM technologies that can exceed the performance of SRAM-based FPGAs in specific areas. However, improving programming latency and endurance is a crucial area of focus.

7.2. Discussion

In our set research question, we were tasked with the development of a hardware structure which can fulfill the three key functionalities of an FPGA. However, in our architectural discussions an important trade-off emerged when combining memory and logic functions within a unified tile. The number of bits in the crossbar array plays a decisive role in determining the LUTs' size and the memory capacity of a single tile in its memory mode configuration.
Increasing the memory size increases data storage capacity within an individual tile. However, this increase presents challenges when tiles are in logic configurations, often resulting in a higher area inefficiency. The optimisation dilemma arises because the enhanced memory footprint impacts the size of LUTs, thereby affecting the overall area efficiency of tiles configured for logic operations.
Another trade-off identified in the architectural discussion is between the number of functionalities added to the periphery of the tile and the size of an individual tile. In our design for instance, we considered the concept of interconnecting multiple memory configured tiles with each other. Without the implementation of this memory-mode specific hardware the number of tiles needed to interconnect tiles would grow significantly.
This brings us to one of the main limitations of this research, which provides an interesting research direction. Combining the key functionalities into a single component gives designers an extra degree of reconfigurability. However, the research method used in this thesis can not say with certainty if unifying resources in a single tile is a better alternative to topologies where resources are fixed. We showed that we can perform better than an SRAM-based FPGA with our micro-benchmark. We showed that

we can outperform the FPGA with a logic-only benchmark, which shows promising results, however a more fair comparison can only be made with more extensive benchmarking.
Comparing more benchmarks was impractical for this thesis due to the lack of tools to implement standard FPGA benchmarks on this design. The micro-benchmark employed in this thesis gives insight into the development of applications for this architecture. And because of earlier work done by other researchers, we can also use the adder benchmark to compare our research to other state-of-the-art architectures. However, as mentioned earlier a more fair comparison would be to benchmark these results against a modern FPGA. These Benchmarks would entail an implementation of thousands of independent logic functions interconnecting with each other. Of which the manual creation of system descriptions which implement complex functions like that is nearly impossible. Computer-aided design tools that can synthesize HDL descriptions to map this architecture will solve this problem. When such a tool is developed, a fair indication can be made about resource utilization and required routing lengths. It is important to know that Liquid Silicon did develop an synthesize tool for their architecture [22]. In that work there were indications that critical paths were reduced due to their flexible topology of resources.

7.3. Main Contributions

In Section 1.2, we presented our research objectives for this thesis. These objectives aimed to build an understanding of the complexities associated with creating an RCA based on emerging eNVM technology. Our primary means of verifying and analysing the objectives is using an event-based simulator created for this thesis. The main contributions of this thesis are listed below:

- Investigation on emerging eNVM technologies in the context of reconfigurable computing architectures.
- Identification of the current state-of-the-art reconfigurable computing architectures based on emerging non-volatile memory technologies which forego mere SRAM-replacement of common FPGA components.
- Creation of a novel FeFET-based reconfigurable computing architecture implementing hardware components which can flexibly and interchangeably operate as functional, routing and memory structures.
- Creation of an event-based simulator. This simulator can verify the novel architecture's functional correctness considering sensing/programming latency and extrapolating energy usage based on the mapped program.
- Evaluation of different emerging non-volatile memory technologies in the context of the novel architecture.
- Evaluating the novel architectures with other state-of-the-art reconfigurable computing architectures based on emerging non-volatile memory technologies.

To address our research question – "How can we create a reconfigurable hardware structure utilizing emerging eNVM technologies that can flexibly and interchangeably operate as functional, routing, and memory structures?" –we introduced a novel RCA and potential implementations of its associated components. We showed some of the structures needed to implement a fine-grain RCA. Furthermore, our architecture performs faster and more efficiently by improving the PDP 2.6X for our 8-bit adder benchmark than the commonly used SRAM-based FPGA. With this work, we ultimately show the potential of eNVM technologies in the development of new RCAs. We expect that integrating these kinds of memory technologies in fine-grain RCA design has the potential to significantly improve the efficiency and performance of these types of architectures.
Furthermore, from our analysis and iterative design process, we have found the following key observations for the development of eNVM-based RCA design:

- **Assess Application Needs**: By profiling the application space the architecture is designated for an appropriate memory technology can be selected. In this work we discussed the differences between the different memory technologies and there influence on different operations. For instance, FeFET is not suitable for applications which execute many write operations at a high frequency, but does offer the best energy efficiency. Choosing the appropriate memory technology depends on the application needs.

- **Design with Look-up Tables**: Incorporate LUTs to simplify logic operations shortens the amount of sequential steps needed to execute an given operation. Although this might necessitate more peripheral circuitry, the overall tile efficiency is optimized.
- **Balance Memory Array Size**: Depending on the application's requirements, strike a balance in the memory array size when switching between memory and logic modes. An appropriate size will ensure large enough data storage in memory mode while still being efficient for logic operations in implementing configurable logic.

7.4. Future Work

7.4.1. Development of a CAD tool

We already discussed the importance of CAD tools to compare this architecture more fairly with other architectures. With CAD tools, we specifically imply software that can translate hardware descriptions from a common HDL like VHDL or Verilog to a hardware description for this architecture. A CAD tool for this architecture would consist of a couple of steps, and some are very akin to the steps used in tools for FPGAs. In the following list, we present a rough outline of the sequential steps needed to synthesize designs from HDLs to a hardware mapping for this architecture:

1. **Conversion:** From an HDL description to an RTL description of the application.
2. **Technology mapping:** Conversion from a boolean network to K-input LUT mapped netlist.
3. **Resource partition:** This step is unique to architectures like this as it determines the amount of memory, interconnect and logic resources needed.
4. **Topology mapping:** Determines the topology of the tiles grid, thus configuring which tile is in logic, interconnect of memory mode.
5. **Placement & Routing:** Fitting the netlist, then configuring topology and each tile's connections.
6. **Optimization:** Applying optimization strategies to reduce the number of utilized tiles or to reduce the critical paths.

The resource partition and topology mapping are unique to this architecture and pose some challenges. One of those challenges is the relation between the mapping of the topology and the routing of the LUT, memory and IO with each other. The most efficient reduces the number of tiles configured in interconnect mode and uses the interconnections of the tiles.
The most area-efficient designs entirely use the available memory capacity of the crossbar array of each tile. Area-wise, mapping a 2-input LUT (4 bits) on a logic-configured tile is less efficient, which can implement a 6-input LUT (64 bits). One of the ways to create more efficient designs is to implement multiple distinct functions on the same tile, which are used at different times in the execution of an application. To reach this kind of efficiency, the CAD tool must cluster logic functions and signal paths to 'recycle' them, which would also be challenging.
In conclusion, adding a complete CAD tools suite is required to use this or similar designs for real-world problems and more extensive benchmarking of this design. Such a tool can be integrated into workflows like VTR [21], but they need additional steps in the synthesizing process to apply to this architecture.

7.4.2. Optimizations

There are still a couple of optimizations which can be applied to the presented architecture in this thesis. The first avenue are the circuits we presented in Section 4.3 which are viable options to implement for this architecture. However, we did not use extensive optimization strategies for these circuits. For instance, the transistor scaling and positioning are based on the designs of previous work. Further circuit-level optimisations would improve power usage and area utilization. Future work can concentrate on creating more efficient circuits for the components specified in this architecture.

The tile size is another avenue for optimization. The tile size can be optimized in numerous way, in this work we explored the we solely used the 8x8 crossbar array size. Further research can be done by changing those array sizes and remapping the benchmarks to accommodate those different tile formats. In combination with a CAD tool optimized tile formats for groups of applications can be determined. There is great dependency of the amount of tiles used, the size of the crossbar array and the way an application is mapped. Thus optimizing the area size would also be an area of interest.

References

[1] Andrew Boutros and Vaughn Betz. “FPGA Architecture: Principles and Progression”. In: *IEEE Circuits and Systems Magazine* 21 (2 May 2021), pp. 4–29. ISSN: 1531-636X. DOI: 10.1109/MCAS.2021.3071607. URL: https://ieeexplore.ieee.org/document/9439568/.

[2] T. Tuan and B. Lai. “Leakage power analysis of a 90nm FPGA”. In: IEEE, pp. 57–60. ISBN: 0-7803-7842-3. DOI: 10.1109/CICC.2003.1249359.

[3] An Chen. “A review of emerging non-volatile memory (NVM) technologies and applications”. In: *Solid-State Electronics* 125 (Nov. 2016), pp. 25–38. ISSN: 00381101. DOI: 10.1016/j.sse.2016.07.006. URL: https://linkinghub.elsevier.com/retrieve/pii/S0038110116300867.

[4] Hiroyuki Akinaga and Hisashi Shima. “Resistive Random Access Memory (ReRAM) Based on Metal Oxides”. In: *Proceedings of the IEEE* 98.12 (2010), pp. 2237–2251. DOI: 10.1109/JPROC.2010.2070830.

[5] Yangyin Chen. “ReRAM: History, Status, and Future”. In: *IEEE Transactions on Electron Devices* 67.4 (2020), pp. 1420–1433. DOI: 10.1109/TED.2019.2961505.

[6] Shimeng Yu and Pai-Yu Chen. “Emerging Memory Technologies: Recent Trends and Prospects”. In: *IEEE Solid-State Circuits Magazine* 8.2 (2016), pp. 43–56. DOI: 10.1109/MSSC.2016.2546199.

[7] Furqan Zahoor, Tun Zainal Azni Zulkifli, and Farooq Ahmad Khanday. “Resistive Random Access Memory (RRAM): an Overview of Materials, Switching Mechanism, Performance, Multilevel Cell (mlc) Storage, Modeling, and Applications”. In: *Nanoscale Research Letters* 15 (1 Dec. 2020), p. 90. ISSN: 1556-276X. DOI: 10.1186/s11671-020-03299-9.

[8] Sven Beyer et al. “FeFET: A versatile CMOS compatible device with game-changing potential”. In: *2020 IEEE International Memory Workshop (IMW)*. 2020, pp. 1–4. DOI: 10.1109/IMW48823.2020.9108150.

[9] Kai Ni et al. “A Circuit Compatible Accurate Compact Model for Ferroelectric-FETs”. In: *2018 IEEE Symposium on VLSI Technology*. 2018, pp. 131–132. DOI: 10.1109/VLSIT.2018.8510622.

[10] Xunzhao Yin et al. “Exploiting ferroelectric FETs for low-power non-volatile logic-in-memory circuits”. In: ACM, Nov. 2016, pp. 1–8. ISBN: 9781450344661. DOI: 10.1145/2966986.2967037.

[11] Ping Chi et al. “Architecture design with STT-RAM: Opportunities and challenges”. In: *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*. 2016, pp. 109–114. DOI: 10.1109/ASPDAC.2016.7427997.

[12] Daisuke Suzuki and Takahiro Hanyu. “Design of an MTJ-based nonvolatile lookup table circuit using an energy-efficient single-ended logic-in-memory structure”. In: *2015 IEEE 58th International Midwest Symposium on Circuits and Systems (MWSCAS)*. 2015, pp. 1–4. DOI: 10.1109/MWSCAS.2015.7282195.

[13] Mengwei Si et al. “Overview and outlook of emerging non-volatile memories”. In: *MRS Bulletin* 46 (10 Oct. 2021), pp. 946–958. ISSN: 0883-7694. DOI: 10.1557/s43577-021-00204-2. URL: https://link.springer.com/10.1557/s43577-021-00204-2.

[14] Lillian Pentecost et al. “NVMExplorer: A Framework for Cross-Stack Comparisons of Embedded Non-Volatile Memories”. In: IEEE, Apr. 2022, pp. 938–956. ISBN: 978-1-6654-2027-3. DOI: 10.1109/HPCA53966.2022.00073. URL: https://ieeexplore.ieee.org/document/9773239/.

[15] Chong-Myeong Song and Hyuk-Jun Kwon. “Ferroelectrics Based on HfO2 Film”. In: *Electronics* 10 (22 Nov. 2021), p. 2759. ISSN: 2079-9292. DOI: 10.3390/electronics10222759. URL: https://www.mdpi.com/2079-9292/10/22/2759.

[16] Tommaso Zanotti. “Reliability and Prospects of Logic-in-Memory Circuits”. In: IEEE, Mar. 2022, pp. 369–371. ISBN: 978-1-6654-2178-2. DOI: 10.1109/EDTM53872.2022.9798047.

[17] Xiaoming Chen et al. “Power and Area Efficient FPGA Building Blocks Based on Ferroelectric FETs”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 66 (5 May 2019), pp. 1780–1793. ISSN: 1549-8328. DOI: 10.1109/TCSI.2018.2874880. URL: https://ieeexplore.ieee.org/document/8515252/.

[18] Mohammed Affan Zidan et al. “Memristor-based memory: The sneak paths problem and solutions”. In: *Microelectronics Journal* 44 (2 Feb. 2013), pp. 176–183. ISSN: 0026-2692. DOI: 10.1016/J.MEJO.2012.10.001.

[19] Jamie Teunissen. *Exploration of Reconfigurable Computing Architectures based on Processing-in-Memory Techniques*. Delft University of Technology, 2022.

[20] Yue Zha, Etienne Nowak, and Jing Li. “Liquid Silicon: A Nonvolatile Fully Programmable Processing-in-Memory Processor With Monolithically Integrated ReRAM”. In: *IEEE Journal of Solid-State Circuits* 55.4 (2020), pp. 908–919. DOI: 10.1109/JSSC.2019.2963005.

[21] *Verilog to Routing*. URL: https://verilogtorouting.org/.

[22] Yue Zha and Jing Li. “Liquid Silicon”. In: ACM, Feb. 2018, pp. 51–60. ISBN: 9781450356145. DOI: 10.1145/3174243.3174244. URL: https://dl.acm.org/doi/10.1145/3174243.3174244.

[23] Xiaoming Chen et al. “Merging Everything (ME)”. In: ACM, June 2019, pp. 1–2. ISBN: 9781450367257. DOI: 10.1145/3316781.3322477. URL: https://dl.acm.org/doi/10.1145/3316781.3322477.

[24] Behnam Khaleghi and Hossein Asadi. “A Resistive RAM-Based FPGA Architecture Equipped With Efficient Programming Circuitry”. In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 65.7 (2018), pp. 2196–2209. DOI: 10.1109/TCSI.2017.2778113.

[25] Mingyen Lee et al. “FeFET-Based Low-Power Bitwise Logic-in-Memory with Direct Write-Back and Data-Adaptive Dynamic Sensing Interface”. In: *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*. ISLPED ’20. Boston, Massachusetts: Association for Computing Machinery, 2020, pp. 127–132. ISBN: 9781450370530. DOI: 10.1145/3370748.3406572.

[26] Xiaoyong Xue et al. “Low-Power Variation-Tolerant Nonvolatile Lookup Table Design”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 24.3 (2016), pp. 1174–1178. DOI: 10.1109/TVLSI.2015.2426876.

[27] Saeyang Yang. *Logic synthesis and optimization benchmarks user guide: version 3.0*. 1991.

[28] Kevin E. Murray et al. “Titan: Enabling large and complex benchmarks in academic CAD”. In: *2013 23rd International Conference on Field programmable Logic and Applications*. 2013, pp. 1–8. DOI: 10.1109/FPL.2013.6645503.

[29] Kevin E. Murray et al. “VTR 8: High Performance CAD and Customizable FPGA Architecture Modelling”. In: *ACM Trans. Reconfigurable Technol. Syst.* (2020).

[30] Accellera. *SystemC 2.3.4 (Includes TLM)*. URL: https://www.accellera.org/downloads/standards/systemc.

[31] “IEEE Standard for Standard SystemC Language Reference Manual”. In: *IEEE Std 1666-2011 (Revision of IEEE Std 1666-2005)* (2012), pp. 1–638. DOI: 10.1109/IEEESTD.2012.6134619.

[32] Hu Chenming et al. *BSIM4.3.0 MOSFET Model*.

[33] Hansraj, Alka Chaudhary, and Ajay Rana. “Ultra Low power SRAM Cell for High Speed Applications using 90nm CMOS Technology”. In: IEEE, June 2020, pp. 1107–1109. ISBN: 978-1-7281-7016-9. DOI: 10.1109/ICRITO48877.2020.9197869.

[34] Mehrdad Amirkhan Dehkordi and Mehdi Dolatshahi. “An energy-efficient, 6 GS/s dynamic comparator in 90 nm CMOS technology”. In: *Analog Integrated Circuits and Signal Processing* 101 (2 Nov. 2019), pp. 319–330. ISSN: 0925-1030. DOI: 10.1007/s10470-019-01526-7.

[35] Huan-Lin Chang et al. “Physical mechanism of HfO2-based bipolar resistive random access memory”. In: *Proceedings of 2011 International Symposium on VLSI Technology, Systems and Applications*. 2011, pp. 1–2. DOI: 10.1109/VTSA.2011.5872253.

[36] John Reuben et al. “Memristive logic: A framework for evaluation and comparison”. In: *2017 27th International Symposium on Power and Timing Modeling, Optimization and Simulation (PATMOS)*. 2017, pp. 1–8. DOI: 10.1109/PATMOS.2017.8106959.

[37] Shahar Kvatinsky et al. “Memristor-Based Material Implication (IMPLY) Logic: Design Principles and Methodologies”. In: *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 22.10 (2014), pp. 2054–2066. DOI: 10.1109/TVLSI.2013.2282132.

[38] Nishil Talati et al. “Logic Design Within Memristive Memories Using Memristor-Aided loGIC (MAGIC)”. In: *IEEE Transactions on Nanotechnology* 15.4 (2016), pp. 635–650. DOI: 10.1109/TNANO.2016.2570248.

A

Detailed device counts

Table A.1: Device count for this work

	CMOS	FeFET
Grid	64	64
SA	112	0
Selection	172	0
Out Logic	198	80
In Logic	56	0
IO	88	32
Total	602	144

Table A.2: Device count for Merging Everything

	CMOS	ReRAM
Grid	256	256
SA	224	0
Selection Grid	517	48
Out logic	664	32
In logic	29	32
Internal Routing	3414	2048
IO	256	64
Total	5360	2480

Table A.3: Device count for Liquid Silicon

	CMOS	ReRAM
Grid	0	256
SA	832	0
Driver	192	64
T Gate	128	0
CDI	64	128
C-Mem	768	128
Total	1984	576

B

Detailed schematics and operation

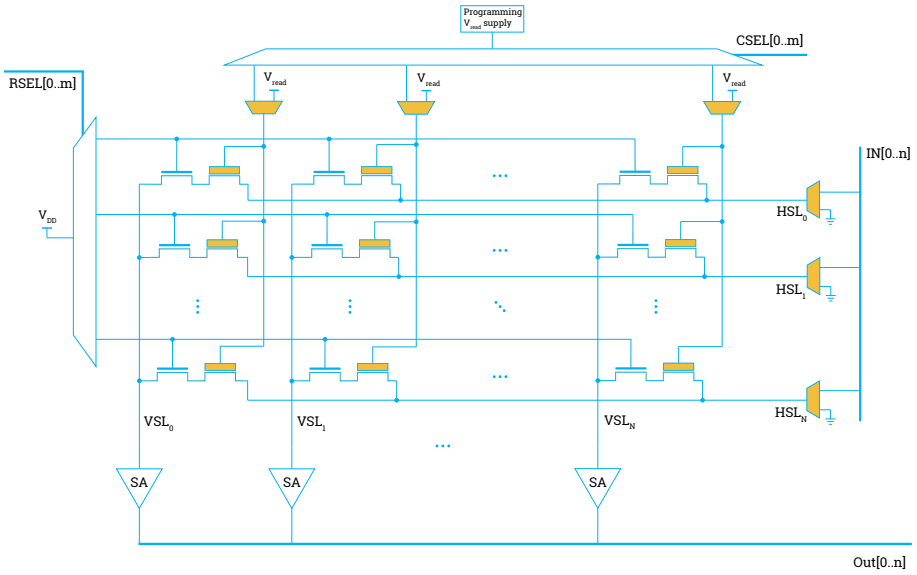


Figure B.1: Detailed schematic of Unified Tile

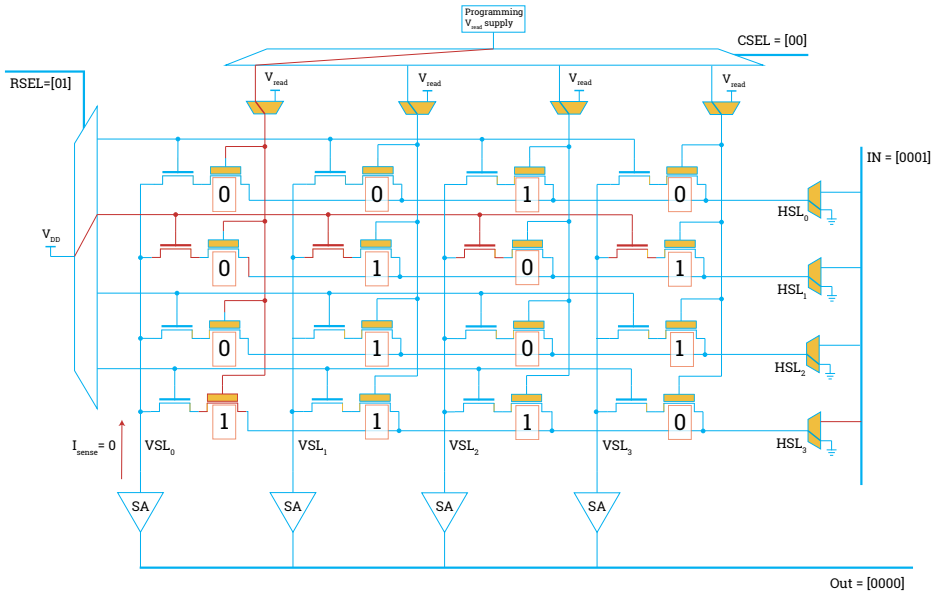


Figure B.2: Detailed Logic operation in Unified Tile

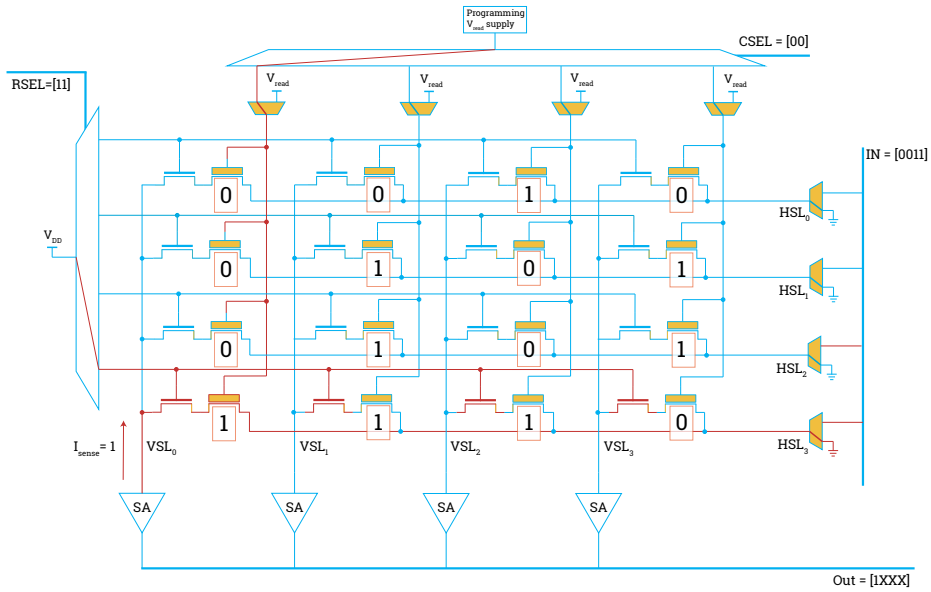


Figure B.3: Detailed Memory operation in Unified Tile