Power Estimation of In-memory Compute based Memory Arrays using Simulink Blocks

A Project Report

submitted by

MOHD. AAMIR

in partial fulfilment of the requirements
for the award of the degree of

MASTER OF TECHNOLOGY

ELECTRONICS AND COMMUNICATION ENGINEERING
INDRAPRASTHA INSTITUTE OF INFORMATION TECHNOLOGY DELHI

NEW DELHI- 110020

3rd October 2022

THESIS CERTIFICATE

This is to certify that the thesis titled "**Power estimation of In-memory compute based Memory arrays using Simulink blocks**", submitted by **Mohd. Aamir**, to the Indraprastha Institute of Information Technology, Delhi, for the award of the degree of **Masters of technology**, is a bona fide record of the research work done by him under our supervision. The contents of this thesis, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Anuj Grover
Thesis Supervisor
Associate Professor
Department of Electronics and
Communication
IIIT Delhi, 110020

Place: New Delhi

Date: 3rd October 2022

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my thesis advisor Dr. Anuj Grover for offering advice and encouragement with a perfect blend of insight and humor. I would also like to thank Belal Iqbal sir, for clarifying my doubts and helping me through different challenging stages of the project.

Special thanks to my friend Somya and my senior Syed Asrar sir for their constant moral support throughout this thesis.

Most importantly, I am grateful for my family's unconditional and loving support.

ABSTRACT

System-on-Chip (SoC) incorporates all the components of any system onto a single integrated circuit. Their compact, power-efficient architecture allows them to be used in various applications like automotive, consumer electronics, aerospace, defense, healthcare, etc. Currently, the architectures are based on separating storage and computing elements. This causes various problems known as a von-Neumann bottleneck and the power wall due to unnecessary data movement between memory and computing units. This causes a lot of power consumption and a reduction in throughput. Since almost 90% of the area is occupied by memory in a typical SoC [1], In-Memory-Computing (IMC) based architectures are being explored as an alternative to overcome the von-Neumann bottleneck problem. This, however, comes at the cost of an increased power budget for memory elements. Hence, it is of utmost importance for a designer to analyze the power consumption of any IMC-based circuit beforehand so as to design an energy-efficient architecture. An IMC architecture contains various elements like bitcells, sense amplifiers, DACs, etc. Each of these elements can have multiple different types of circuit implementations. So, to get the most energy-efficient circuit, a designer should analyze the different designs of such circuits. Existing simulators for power estimation of IMC memory arrays are highly complex and require high design efforts to change the circuit and simulate alternatives. This acts as a bottleneck in proper design space exploration. This calls for an energy estimator, enabling a designer to get a rough estimate of power consumption with reduced design efforts and faster design space exploration.

KEYWORDS: Memory Arrays;Power Estimation; Simulink Models; In-Memory Compute; SRAM

TABLE OF CONTENTS

CHAPTER 1

Motivation

Recently, with the rapid growth in the Internet of Things (IoT), there has been a sudden increase in the amount of data generated and algorithms for processing this data. Most of the IoT devices in the market are now required to run data-intensive machine learning, neural network, or cryptographic algorithms at their core. Since these algorithms process loads of data, memories are a significant factor in the overall performance of these architectures. The modern processors now run at much higher speeds than at which the memories can provide the data. This limits the maximum speed at which the von-Neumann architecture-based system can function, referred to as the von-Neumann bottleneck. Also, the frequent data transfer between processor and memory due to limited data bus-width increases power consumption.

In-memory compute architecture is now being proposed to address these performance and energy issues. Here, the computing units are brought near the memory arrays such that most of the processing can be carried out without any processor's intervention. While IMC architecture can be considered an excellent solution to the von-Neumann bottleneck, it brings new design challenges to the designer.

The memory arrays are built as a collaboration of different sub-circuits like row decoders, bitcell arrays, column decoders, interface, etc., which work together to constitute a working memory. Each block used in the memory array can have multiple circuit implementations possible. These implementations differ in terms of power, performance, and area (PPA). So, any system must undergo extensive simulations with all the possible circuit implementations so the designer can explore the design space to create the system with the most efficient configuration. Hence, the need for an easy-to-use energy simulator with considerable accuracy arises, which is the primary motivation for this work.

CHAPTER 2

Introduction to SRAM Memory Arrays

In any computing system, memory arrays play a central role as they store the data required for computing. A memory array consists of a two-dimensional 2^Mx2^N array of memory bitcells (where M and N are column and row address widths, respectively), with each cell storing a single bit of data. Contents of this array are accessed by asserting the word lines (WLs) of a particular row. The location of this row is called address, and values stored in bitcells of that row are the data required by the user.

An SRAM array is a collaboration of multiple circuits working sequentially to read/store data. An overview of all such sub-circuits of the SRAM memory is shown in Fig. 2.1. These sub-circuits are explained in detail as follows.

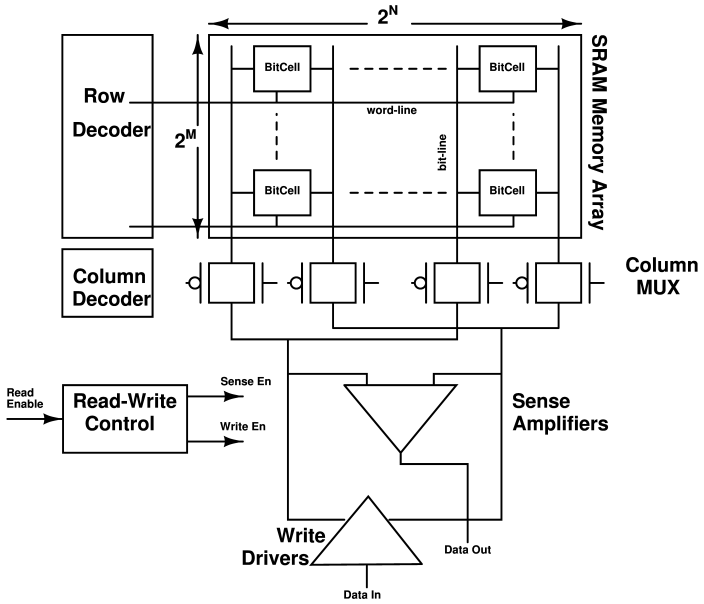


Figure 2.1: SRAM Memory Architecture

2.1 Bit-cells

In an SRAM array, a bitcell is used to store a single bit of data. The primary storage element inside a bitcell is a cross-coupled inverter pair, forming a bi-stable latch. This

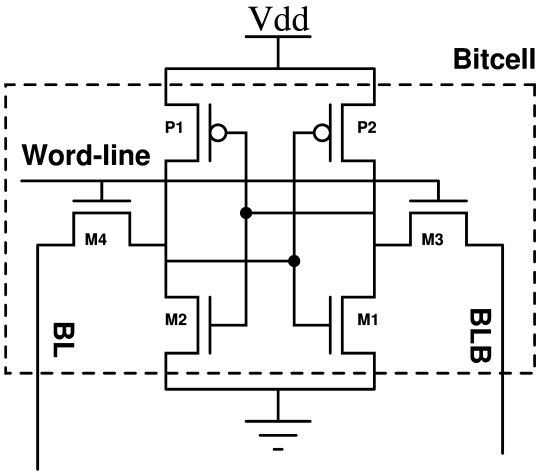


Figure 2.2: 6T Bitcell

latch stores a bit -0 or bit-1 as a pair of complement bits. For read and write operations, the output values of these inverters are connected to bit lines (BL, BLB) via MOS devices. The circuit for a 6T bitcell is shown in Fig. 2.2.

The number of ports in a memory array depends on the architecture of the bitcell used. In a single port memory, we have a single differential pair of bitlines for read and write operation. The Fig. 2.2 shows the most used bitcell circuit called the 6T SRAM cell. In the 6T cell, the inner data point (output of cross-coupled inverters) is connected to bitlines using nmos devices in a pass gate orientation. Some of the other types of bitcell architecture are discussed as follows.

2.1.1 8T SRAM Bitcell

In 6T bitcell, there is a common port for read and write operations. Read operations demands a better cell current for faster operations. But, in the 6T cell Fig.2.2, because internal storage nodes are directly connected to bitlines via pass transistor, stability issues (cell flipping errors) may occur if the pass transistor(M3,M4) in Fig.2.2 is made strong. Hence, for better readability, without compromising on the stability of the cell, 8T bitcells are used. In 8T cells, a separate read port is isolated from internal nodes via gate capacitance. Since the read port does not directly impact internal nodes, designers can independently size the read port nmos'es (M5, M6). The 8T cell is shown in the fig 2.3.

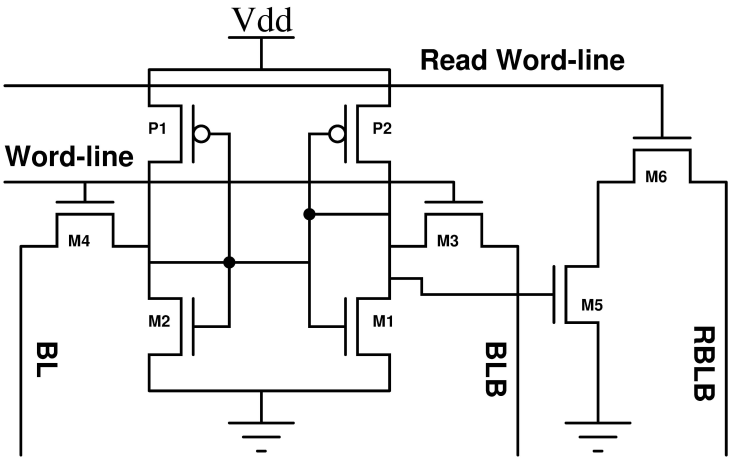


Figure 2.3: 8T Bitcell

2.1.2 10T SRAM Bitcell

The 10T bitcell, similar to the 8T cell, provides a separate independent read port at the cost of two extra nmos devices. The difference between 8T and 10T is that a 10T cell provides two single-ended read port capabilities. The cell circuit is shown in the figure 2.4.

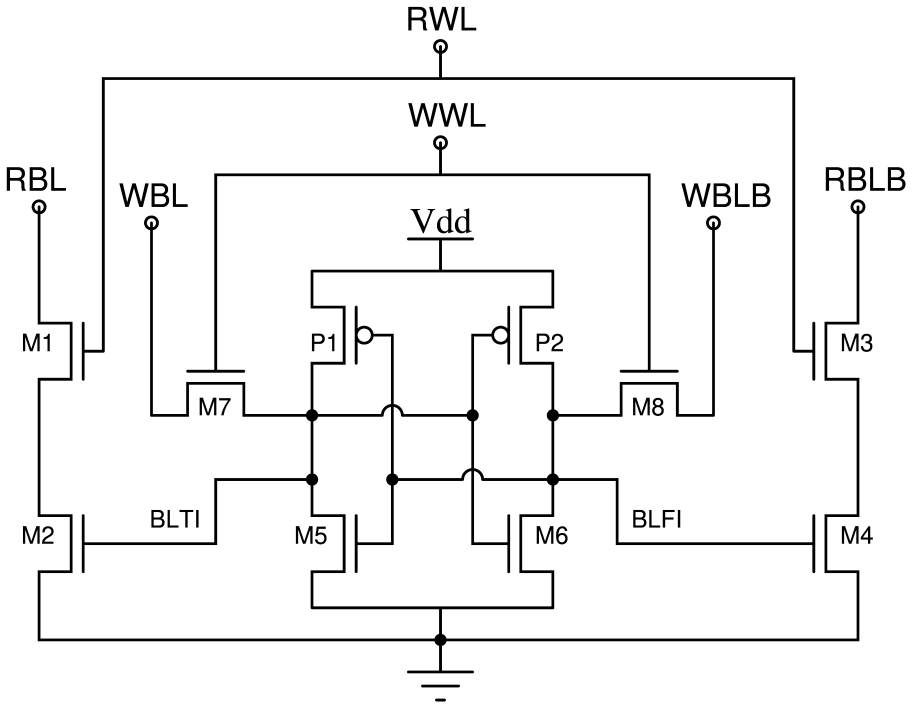


Figure 2.4: 10T Bitcell

2.2 Decoders

An SRAM bitcell is accessed for read/write operation by its respective row and column. For this, row and column decoders are attached to the memory array which maps the input address bits to the particular row and column in the memory array.

2.2.1 Row Decoders

The path from the input address lines to the word-line constitutes of a circuit called as row decoder. These row decoders are the logic circuits that map N-bit input address to 2^N word-lines. The circuits is divided into two parts, i.e., Pre-Decoder stage and Post-Decoder stage. Structure of a row decoder is shown in Fig. 2.5.

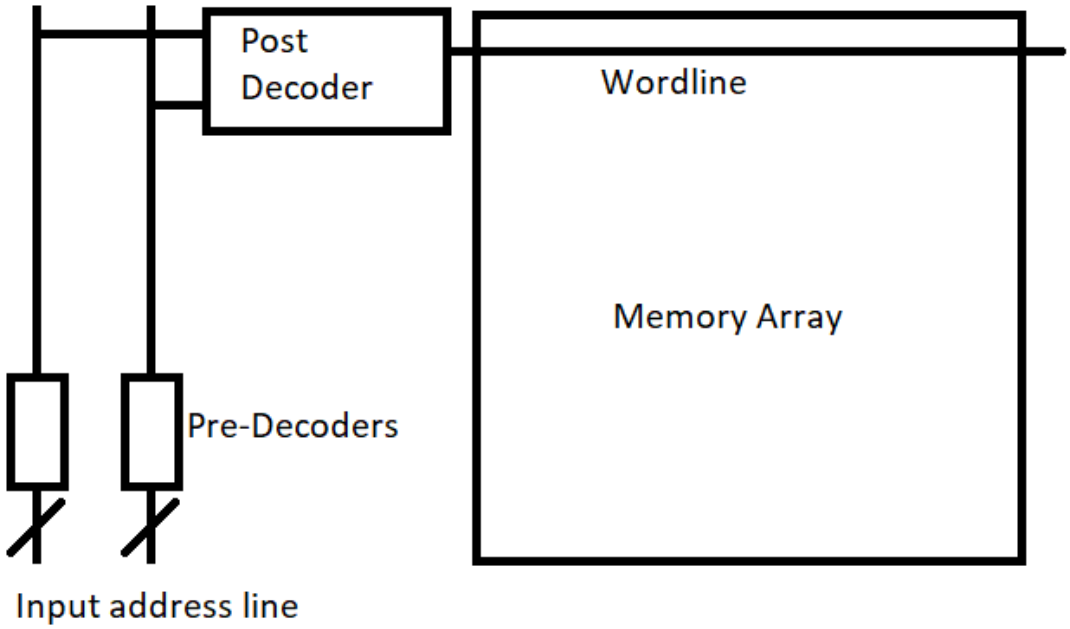


Figure 2.5: Row-Decoder Architecture

2.2.2 Column Decoders

In bitcell arrays, a squarish form factor is preferred over a long rectangular one. It is because when the write operation is carried out, there is a race between two paths, i.e., input address to word-line access path and read-write control signals, write driver to bitline conditioning path. The delays are equally spread in a squarish array to avoid any delay-related faults.

In memory arrays, we use rows to store a single word and row decoders to access that. However, since we need to save a large amount of data, storing a single data element in each row would lead to a long rectangular array. So, to achieve a squarish form factor, multiple data elements are stored in a single row. Then, column decoders implemented using column mux are employed at the IO circuits to decode the required data elements from a single row.

2.3 Input-Output (IO) Circuitry

The memory arrays need IO circuitry to interact with the external environment. It acts as the access point to read from and write to the inner bitcell array. The IO circuitry consists of the following discussed blocks.

2.3.1 Sense Amplifiers

In an array, each bit of data is stored in a single bit-cell. Since memory arrays are used to store huge amounts of data, to save area, bitcells are made as small as possible. However, this comes at the cost of very weak discharge of the bitlines and considerable delays in the read operation. Additionally, discharging bitlines completely in each read cycle would lead to huge power consumption. So, sense amplifier circuits (SA) are added at the IO periphery which are used in the read operation. Sense amps are used to detect the potential difference of orders of mili-volts between bitline and bitline-bar, which is then amplified to output as 0 and 1. Hence, while reading from a bitcell, the bitlines are not required to be discharged entirely, thus saving power and latency.

2.3.2 Write Drivers

Write drivers are part of the IO circuitry used in the writing operation. During the write operation, drivers are used to pre-condition the bitlines to either Vdd or ground depending on the bit value to be written. When the word-line is asserted, based on the bit line's pre-conditioned value, the respective bitcells are flipped. These are mostly a series of buffers connected to column decoders.

2.4 Operations in Memory Array

2.5 Read Operation

The bitlines are initially charged to voltages close to Vdd in a read operation. Then, the pre-charge circuit is disabled, and bitlines are left floating. When the word-line is asserted, the bitline connected to the node storing a “0” bit is discharged. In contrast, the complementary bitline, i.e., bitline-bar (in the case of differential bitlines), remains at the pre-charged state. This differential discharge generates a potential difference between the bitlines. The sense amplifier then uses this offset to detect a “1” or a “0” value stored in bitcell. Here, the power consumption directly depends on how long the bitlines are being discharged since the pre-charge circuit will have to charge the bitlines in the next cycle by drawing power from the source.

2.6 Write Operation

During a write operation, the write data is fed to the bitcells with the help of differential bitlines pair. The bitline and bitline-bar of the required columns are pre-conditioned with write drivers. If the target cell has bits in an opposite pattern to bitlines potential, then the internal node storing a “1” is discharged via the access transistor. This causes the data to be flipped in the cell. Hence the cell undergoes a write operation. It should be noted that, unlike the read operation, bitlines in the write operation are actively driven by the write driver.

CHAPTER 3

In-Memory Compute Architectures

Due to the increased usage of intelligent wireless devices, demand of energy-efficient and high-speed architecture have increased. Driven by Moore's law, a steady increase in the performance offered by the computing cores has been observed. However, there has been no such steady improvement in memory technologies. The computing system is divided into processing and storage blocks in a conventional von-Neumann architecture. So, even if the cores are efficient enough to run at high speeds, they are limited by the data access time from memories. Secondly, with respect to the power consumption, we know that machine learning, neural network, and graphical data processing are a family of the most used algorithms in any smart device. These algorithms are data-intensive. Hence there is a lot of data movement between computing cores and memories, causing unnecessary power consumption and performance bottleneck due to data-bus width. To overcome the problems of power consumption and performance overheads in conventional architectures, IMC architectures are proposed. Here, the computations are executed directly inside the memory array instead of first moving the data out of the array before computing. Multiply–accumulate (MAC) operations and memory accesses are the two most dominating operations in neural networks. Many IMC architectures are proposed capable of executing these operations using the memory arrays themselves [2],[3]. IMC architecture is basically distributed in two sections, the first is the bitline based IMC operations and the other is the Near-Memory Compute (NMC) based operations.

3.1 Bitline-based IMC Operation

The concept of In-memory computing, where the bit-cells are used to compute operations like MAC, depends on the process of bit-lines shorting. Here, wordlines of multiple rows are asserted at the same time, and this causes multiple bitcells in a single column to get activated simultaneously and contribute to cell current. The cell current

contribution from each bitcell depends on the cell's data and the duration of word line pulse width. The final cell current which discharges the bit-line is the sum of all these individual bitcell currents.

Based on data stored in each bitcell and durations for which each word line is asserted, we get different amounts of bitline potential discharge (VBL). This dependency of bitline potential on bitcell data and wordline pulses is used in MAC operations. Few Papers that take advantage of bitlines shorting for doing operations like XOR, or MAC are [1],[2], [4], [5], and [11].

3.2 Near-Memory Compute (NMC) Elements

An IMC module typically requires additional circuitry apart from the memory array, usually placed at the IO block. Custom circuits are designed and placed near the IO block of the memory for complex arithmetic or boolean operations, which are referred to as NMC elements. For example, in Fig. 3.1, the IMC implementation of ChaCha20 algorithm is shown. Here, the adder and the rotation circuit will be called NMC elements. With this architecture, the entire ChaCha20 algorithm is implemented in the memory array itself. This bypasses multiple transactions between memory and the processor for frequently used operations by implementing them near the memory array. This saves on power consumption and enhances the system's performance. In other works like [4] and [5], capacitor-based weight processors and ADCs are used at IO for generating MAC output.

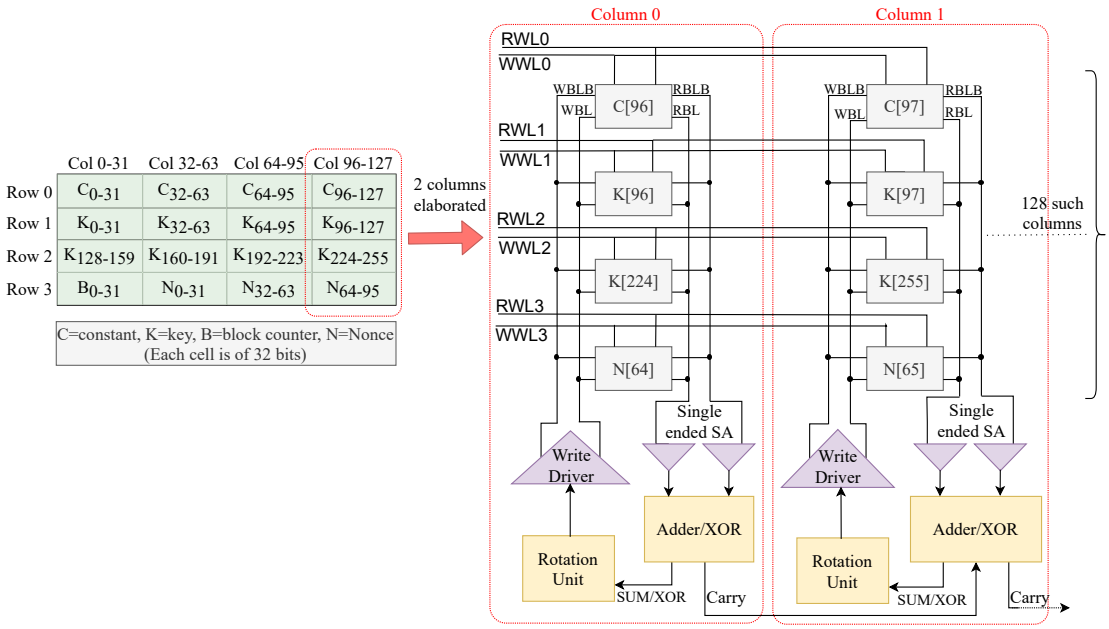


Figure 3.1: In-memory-ChaCha20 Architecture

CHAPTER 4

Simulators for Memory Arrays

Power, performance, and area (PPA) estimates have been crucial aspects of any circuit. The amount of design efforts needed to satisfy the user requirement depends heavily on these early-stage analyses. Also, these estimates and proper design space exploration at early stages lead to faster design flow convergence. For this purpose, circuit simulators are used to evaluate the circuit's functionality and understand the PPA footprints. We use these simulators to emulate how the chip will function in a real-life scenario. Multiple complex circuit simulators, such as Hspice and Eldo, use complex differential equations and multiple higher-order technology parameters to model circuits. These simulators are highly accurate since they model many of the parasitics involved such that the design is as close to the real scenario as possible. Monte-Carlo simulations are one such example of circuits simulation, in which the technology parameters of the devices are varied, and the circuit is re-simulated to incorporate the impact of mismatch due to process variations. Although these simulators provide accurate estimations for PPA, they require significant simulation time and high design efforts in design space exploration.

In an IMC array, multiple blocks work collectively in a timed manner. Circuits for each block can have numerous possible implements, each having its advantages and functions. So, suppose a design moves forward with specific design implementation. If in the later design stage, it is observed that some other architecture could have provided better results regarding power consumption. The entire design flow now has to be rolled back. It becomes a time-consuming and high-effort task for the designer to again go over the circuits, modify them, simulate for worst-case scenarios, configure the design parameters to meet the timing constraint, and redo the physical layout. A simulator can help avoid redoing these circuit designing steps. It can simulate the IMC arrays to give precise estimations for PPA metrics based on the data being processed. Any simulator should be flexible enough to allow the designer to change any of the blocks with minimal design effort. It will allow the designers to run through many architectures and have a good amount of design space exploration to get the most efficient circuit.

Using these simulators at the beginning of the design cycle will save the design efforts and the time to market in designing a particular IMC memory array.

4.1 Literature Review: Existing IMC simulators

Many simulators for IMC-based memory arrays have been proposed, a few of which are discussed. In [6], a simulator is presented that can analyze the application code and recognize the instructions which can be offloaded as IMC circuits. It also creates a high-level abstract model of IMC logic blocks to estimate parameters like computing latency and energy consumption per instruction when the given instruction is assumed to be offloaded to the memory array. Also, without even requiring actual IMC circuit design, users have the flexibility to define any logic or even processor core as an IMC logic module.

Architecture-level simulators, like the one described in [7], are used to compute power and area estimates of the IMC accelerator. The simulators provide users with two levels of templates to represent their design. The first template is to describe the memory architecture at an abstract level. Here, the user needs to put information like data width of different blocks, number of rows and columns, and number of reads and write ports. The second level of templates describes the details of individual components like the ADC and DAC, memory cell, etc. It has details like the data width, CMOS technology, latency, etc., needed to identify a component. Further, these data are sent into energy reference tables (ERTs), which use pre-existing block level estimates. These tables are then used to calculate the energy consumption of the user-defined architecture.

Simulators like [8] use multiple pre-existing modeling tools and simulation tools, such as GEM5, McPAT, and DESTINY, to model and analyze the interaction between CPU and memory arrays. This simulator is used to predict the overall energy consumption and performance of any system that contains an IMC module as a part of computing units. This tool uses an array model for power estimation, which contains power-consumption estimates in terms of individual IMC operations like IMC-XOR operation. These estimates are pre-calculated using Hspice by simulating the circuit netlist. For IMC circuits of which netlist is unavailable, the user can break down the

new IMC operation into multiple atomic operations like bitcell access, sense amp access, write op, etc. These atomic instructions are then passed to power estimation tools like Destiny. This tool has pre-calculated values for each atomic operation and is used to predict the power consumption for the given operation.

Power simulator tool Destiny [9] estimates the power consumption based on device parameters like cell size, supply voltage, reset voltage, etc. Since this tool uses technology parameters to calculate power, it is used for fast design-space exploration across multiple domains like memory technology (SRAM, DRAM, ReRAM, NVM, etc.), device technology (MOSFET, FinFET, etc.) and fabrication methodology (2D or 3D).

In some research, IMC circuits were incorporated with power-saving techniques which relied on the correlation among data bits stored in the memory array. Previous works like [2] have devised specific techniques to save power based on the type of data under process. Even though there were previous IMC architectures on data-dependant processing. Studied simulators lacked data dependency for power estimation since these simulators were based solely on design and technology parameters. We found no IMC simulators to model the power consumption based on the changing inputs of the circuit. Also, many of these simulators had complex templates requiring high design efforts. Moreover, they did not provide enough flexibility to alter the memory array at the granular block-wise level. Further, no previous research was found to have used the MATLAB Simulink tool for modeling and estimating the PPA of IMC memory arrays.

CHAPTER 5

MATLAB Simulink based Power estimation model

This project proposes a MATLAB Simulink-based framework for the IMC memory array. The simulator is used to emulate the functionality of the memory array, IMC operations, and Near-memory compute blocks and give the designer basic estimates of the power consumption. The model provides two types of templates for adding new-block. Firstly, a technology-based template for internal elements, like bitcells, to characterize blocks. It includes fundamental current and voltage equations, leakages, and other characteristic parameters like the number of bitlines. For remaining elements like row decoder, it provides look-up table-based templates. Using complex simulators, LUT-based templates are created by collecting power consumption data at various input vectors. Further, these LUTs are used for power estimation in different blocks. Any new block can be added to the design by providing the data required by these templates. Also, this framework is based add-drop model, in which to add a new design element, the designer has to create the Simulink model and add that to the design. Power in any circuit depends on what data is being processed. For instance, in a static row decoder, output flip from Vss to Vdd has a high power consumption compared to Vdd to Vss, considering the leakages. Therefore, the proposed framework allows users to carry out power consumption of different blocks based on the data they are currently processing.

Finally, this model gives a good overview of all the components, which can be altered and monitored individually. It also allows the user to edit and compose each component at a granular level. Compared to conventional and complex simulators, this simulator can run multiple design cases with different test case data at a comparatively much lower effort.

5.1 Description of model

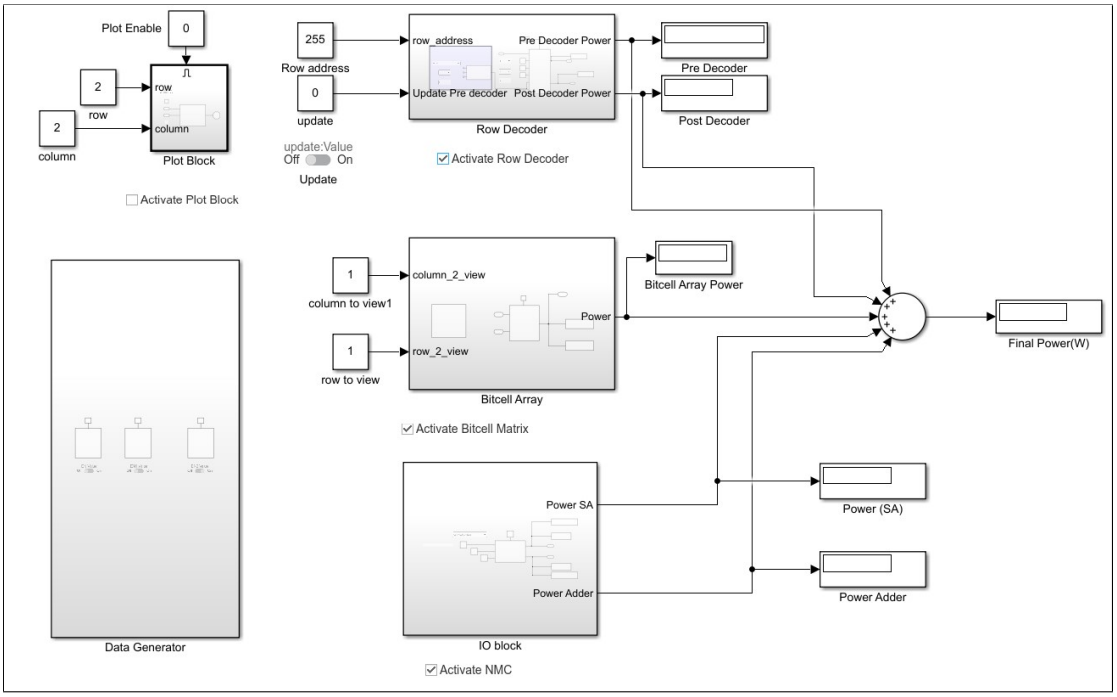


Figure 5.1: Overview: Memory Array in simulink

In figure Fig. 5.1, an overview of the Simulink model is presented, and in Fig. 5.19, the back-end library files used in the simulator are shown. All the different blocks of the architecture are explained in detail as follows.

5.2 Data Generation Block

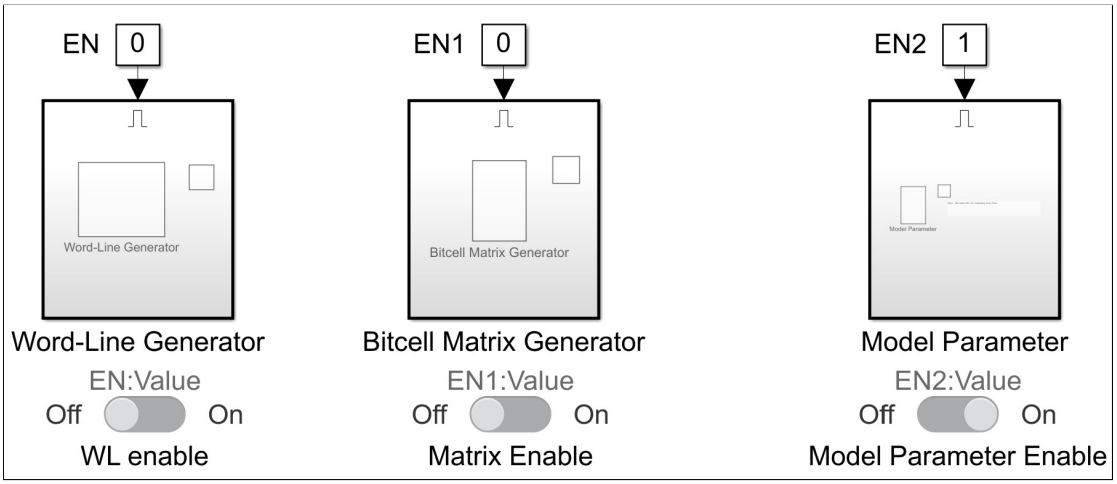


Figure 5.2: Overview: Data Generation Block

The data generation block in Fig. 5.2 has majorly two functions. The first function is to generate test cases for the simulator. Here, test cases are the data stored inside the memory array(weights) and the word-line stimulus(feature data). Secondly, this block allows users to alter the technology parameters of the transistors used in the bitcells. It has three components explained as follows.

5.2.1 Word-Line Generator

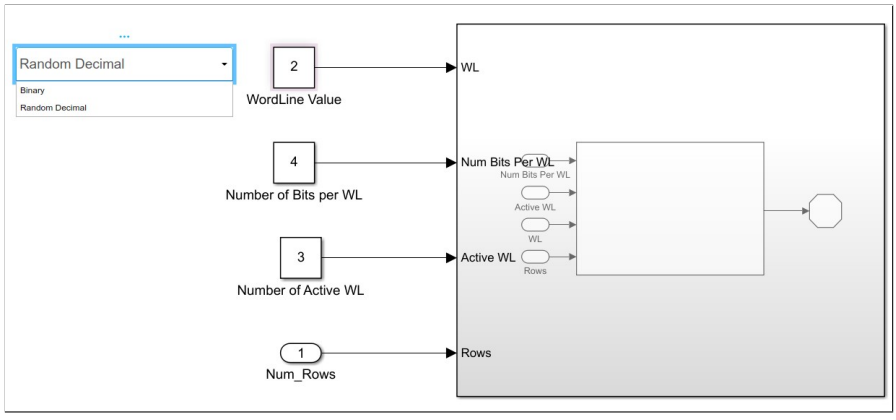


Figure 5.3: Word-Line Generator

A bitcell communicates with the external environment using bitlines, which are accessed via the access transistors. Word lines control these access transistors. When a bitcell is accessed, based on its data content, it discharges the bitlines while the word-line is “ON”. In IMC applications, word-line pulse widths are altered based on input

data(feature data) and used in MAC operations. In our model, this block generates word-line pulse widths vectors for an array containing N-rows.

Users have the choice to either get binary values for word lines (constant pulse width for high word lines) or random decimal values. In the case of random decimal values, non-zero word lines are ON, with pulse width duration as a scalar multiple of a decimal value.

For example, Table 5.1 shows the binary WL vector and decimal WL vector, respectively, for a 6-row bitcell array.

WL	WL
1	7
1	3
0	0
0	0
0	0

Table 5.1: Wordline Types Example

Using these vectors bitcell array block will activate the non-zero Wordline rows for the corresponding duration.

5.2.2 Bitcell Matrix Generator

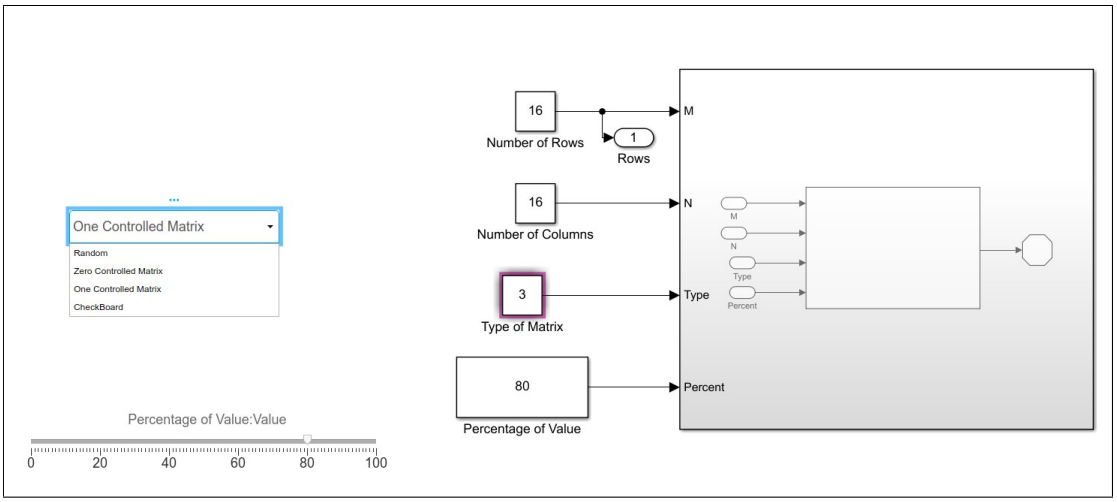


Figure 5.4: Bitcell Matrix Generator

Bitcell arrays are the location where the data bits are stored in the memory. This block generates the data matrix, representing the data stored in the bitcell array. Here it is to be noted that the discharging rate of a bitcell depends on the data stored in the

bitcell of that column. Here, users can select the array size and the data pattern of the matrix. For patterns, the simulator gives users the following four choices.

- Random Matrix: This selection generates a random matrix of 0's and 1's with each cell depicting a bitcell array.
- Zero-dominant Matrix: This selection generates a data matrix, with the number of 0's depending on the parameter "Percentage of value" block in Fig. 5.4.
- One-dominant Matrix: This selection generates a data matrix, with the number of 1's depending on the parameter "Percentage of value" block in Fig. 5.4.
- Checkboard Matrix: This selection generates a data matrix in a checkboard pattern of 0's and 1's.

5.2.3 Model Parameter

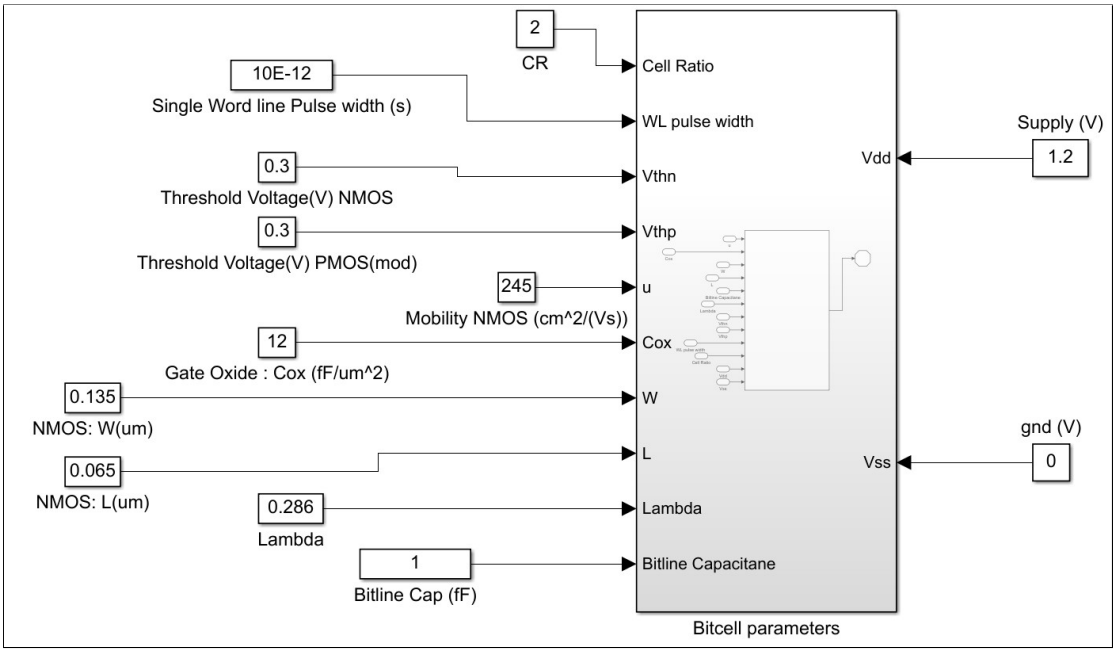


Figure 5.5: Model Parameters

MOS circuits require multiple design parameters for evaluating the mathematical equations required to simulate the transistors. This block allows the user to tweak those technology parameter values. The parameters which can be altered are :

- Mobility (ν)
- Width of MOS (W)
- Length of MOS (L)
- Channel length modulation parameter (λ)

- bitline Capacitance (C_{BL})
- PMOS and NMOS threshold voltages
- Word-line minimum pulse width (P_{WL})
- Supply and sink voltages (Vdd, Vss)

5.3 Row Decoder Block

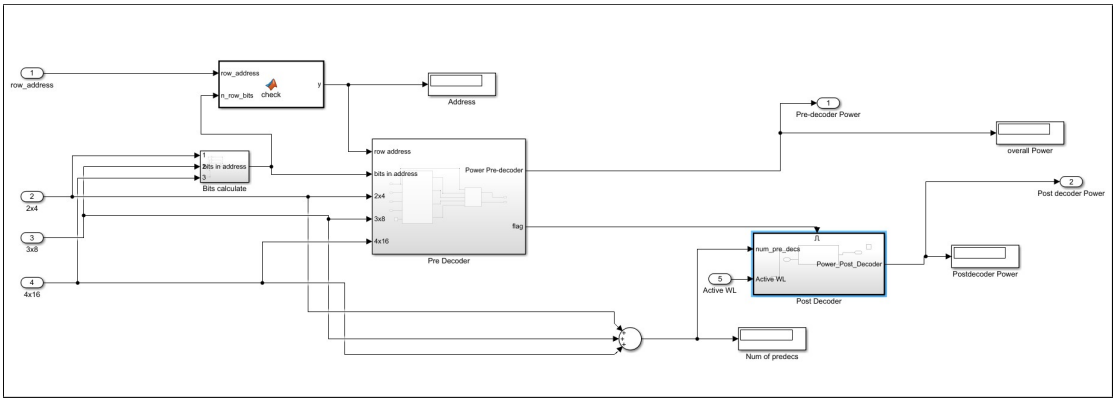


Figure 5.6: Overview: Row Decoder

Row decoder block Fig.5.6 is designed to estimate the power-consumption of row-decoder in a memory array. This block is divided mainly into two sub-blocks. Namely

- Pre-decoder block: This block has two functions. Firstly, it estimates the power consumption of the pre-decoder based on current and previous inputs, and secondly, it converts the input row address into input values for the post-decoder.
- Post-decoder block: Based on the input (for post-decoder) generated from the pre-decoder, this block estimates the power consumption of the post-decoder block.

Apart from these, the row decoder block also contains a few utility blocks like bit-calculate to calculate the number of input lines in the pre-decoder and sanity checks to check if the given row address is correct based on the number of input lines.

5.3.1 Pre-Decoder block

The pre-decoder block includes an initialization block and a power estimation block, which are discussed below.

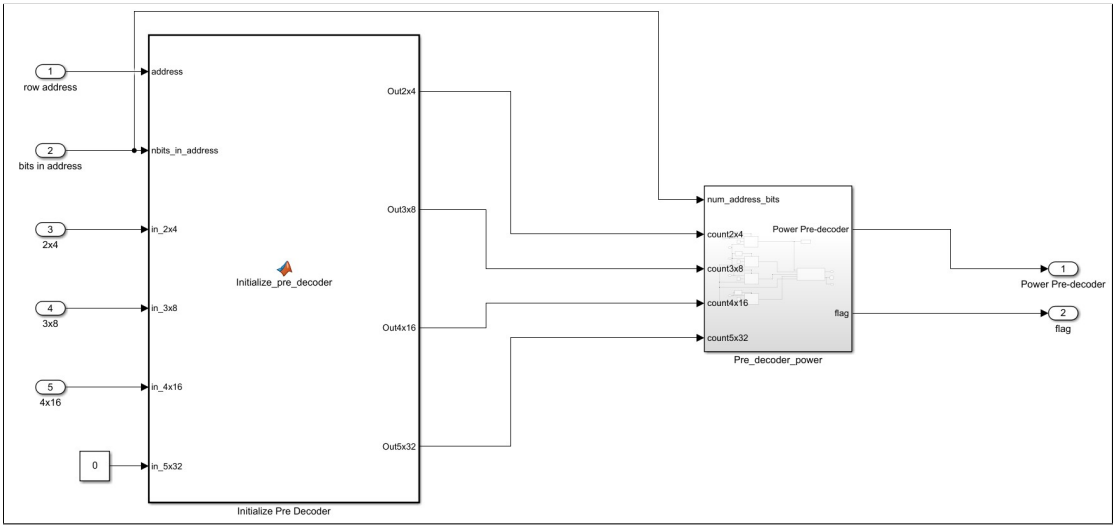


Figure 5.7: Block-1 : Pre-decoder

Initialize Pre-decoder block

This block is used to update the pre-decoder parameter files according to the data input from the user. This block takes the row address, the number of bits in each address, and the number of 2x4, 3x8, and 4x16 pre-decoders used in architecture. It then maps the N-bits row address given by the user to the inputs of each pre-decoder.

Pre-decoder Power block

This block estimates the power consumption for each type of pre-decoder used. As shown in Fig. 5.8, there are four pre-decoder blocks in the first stage. Each of the blocks corresponds to 2x4,3x8,4x16 and 5x32 pre-decoders, respectively. These pre-decoders take the number of each stage used and navigate through the LUT based on current and previous inputs to estimate the power consumption.

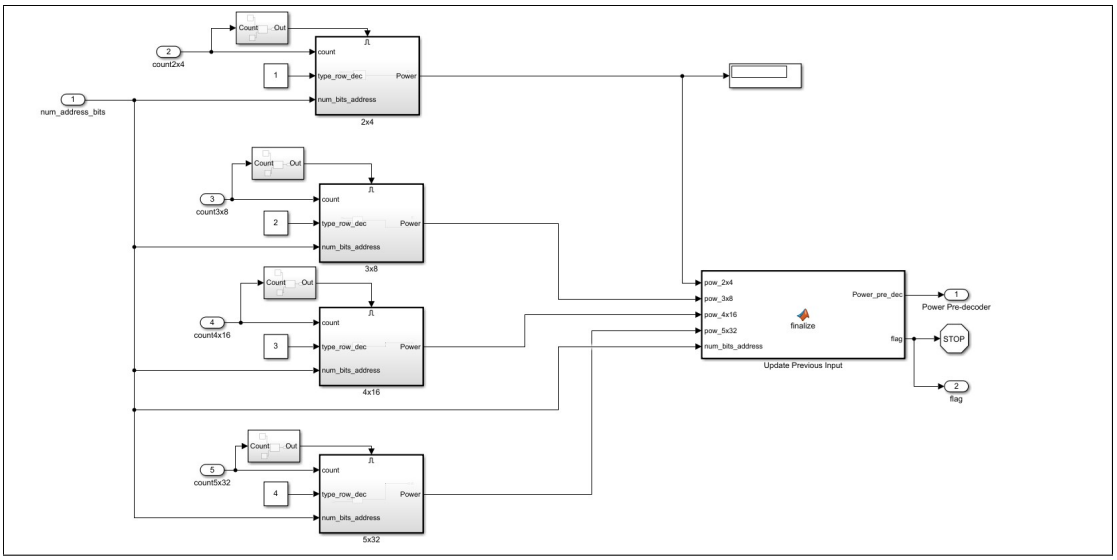


Figure 5.8: Pre-decoder Power block

5.3.2 Post-Decoder block

The post-decoder block includes an initialization block and a power estimation block, which are discussed below.

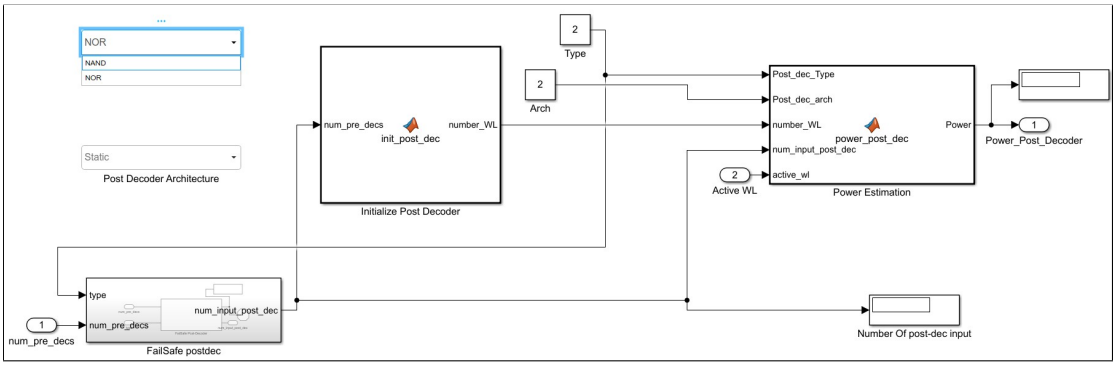


Figure 5.9: Overview: Post-Decoder Block

Initialize Post-decoder block

This block is used to update the post-decoder parameter files according to the data input from the user. and generate the inputs for the post-decoder.

Power Estimation

This block estimates the power consumption for each type of post-decoder used based on the type of post-decoder (NAND or NOR) and the architecture of the post-decoder(static or dynamic).

5.4 Bitcell Array block

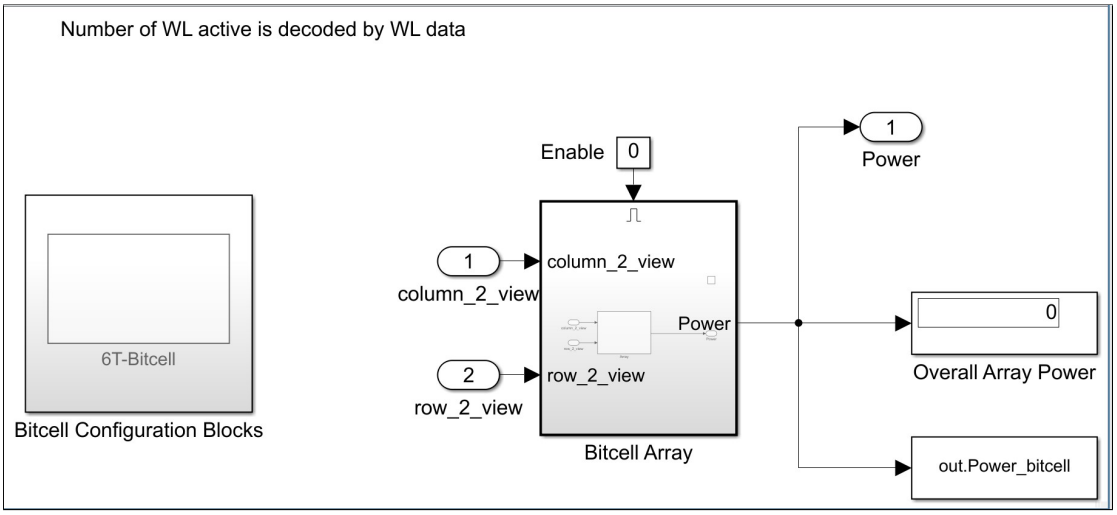


Figure 5.10: Overview: Bitcell simulation model

5.4.1 Introduction

The function of the array block is to model the operation of a bitcell array during bitline-based IMC operations. It emulates the bitlines, word-line, and bitcell interaction during read and IMC operations to estimate the power consumption of the block.

In IMC architecture, bitlines are used for MAC operations. Multiple bitcells in a column are activated for varying durations. The MAC values are calculated as a result of bitcell currents discharging the bitlines. So, monitoring bitline voltages for different test case scenarios is essential as the remaining bitline voltage is converted to MAC output using DACs. Therefore, this block is also designed to plot bitline voltage, WL voltage, bitlines current, and power consumption (per column) graphs.

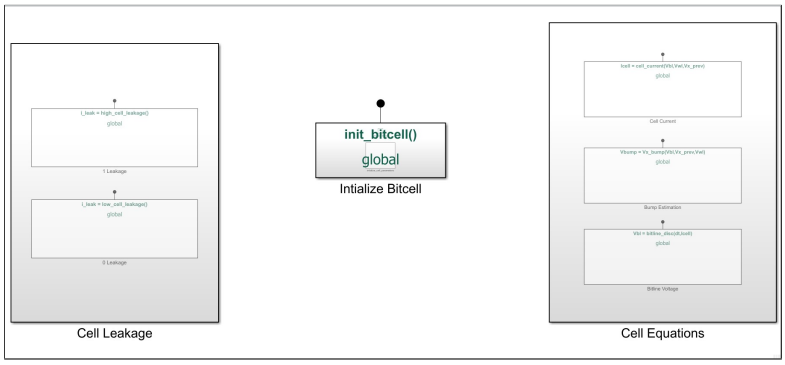


Figure 5.11: 6T bitcell simulation model

5.4.2 Working

Read or IMC operations in a memory array depend on the activated rows, how long each row is activated for, the cell current contribution from each active bitcell, and the leakage contribution. The word-line-related information, like the activated rows, time duration of each WL pulse (IMC feature data), and other design parameters, are extracted from the library files. For bitcell modeling, a cell configuration block is added to the model as shown in Figure 5.11. It contains three sub-blocks, namely.

- Initialization block: This block contains information about the type of bitlines in the cell. For example, a 6T cell has differential pair of bitlines, but an 8T cell has a single-ended bitline.
- Cell Leakage block: This block is used to add information related to the cell leakage current.
- Cell Equations block: This block contains all the necessary functions required to model the read operation in a bitcell.

Further, for the modeling of the power consumption, the block also takes the MOS parameters as input from the user. Finally, all the parameters and equations are combined in the array block to give power estimates and graphs. In Fig 5.12, parameters that can be altered for a bitcell are shown.

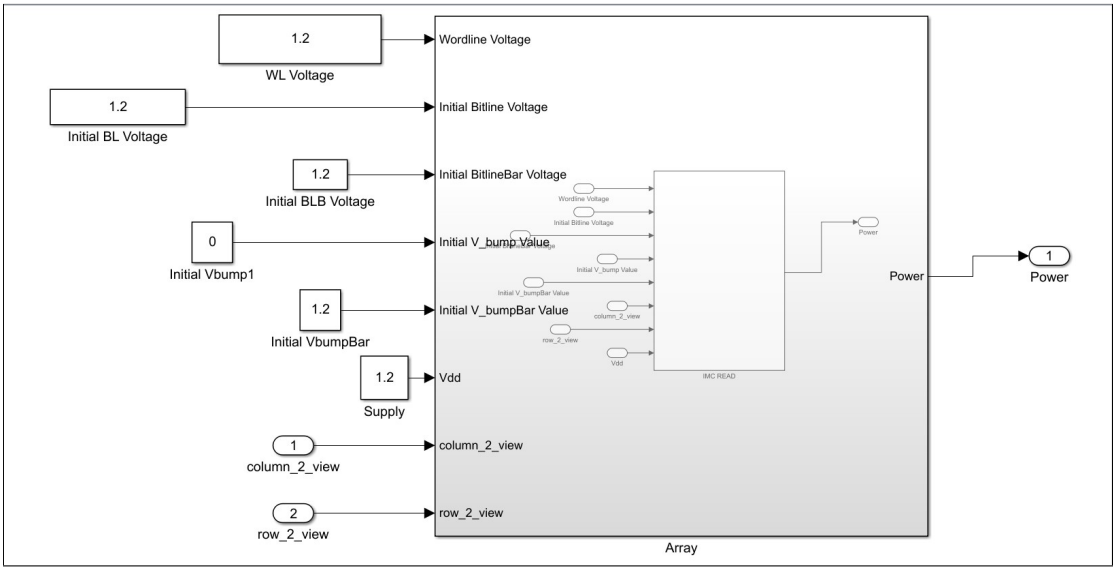


Figure 5.12: Overview: Bitcell simulation model

5.5 Input Output block

IO block, as shown in Fig. 5.13, is used to model the near memory compute element and the IO block of the memory array. It contains two sub-blocks, viz. sense amplifier block and adder block, which are discussed below.

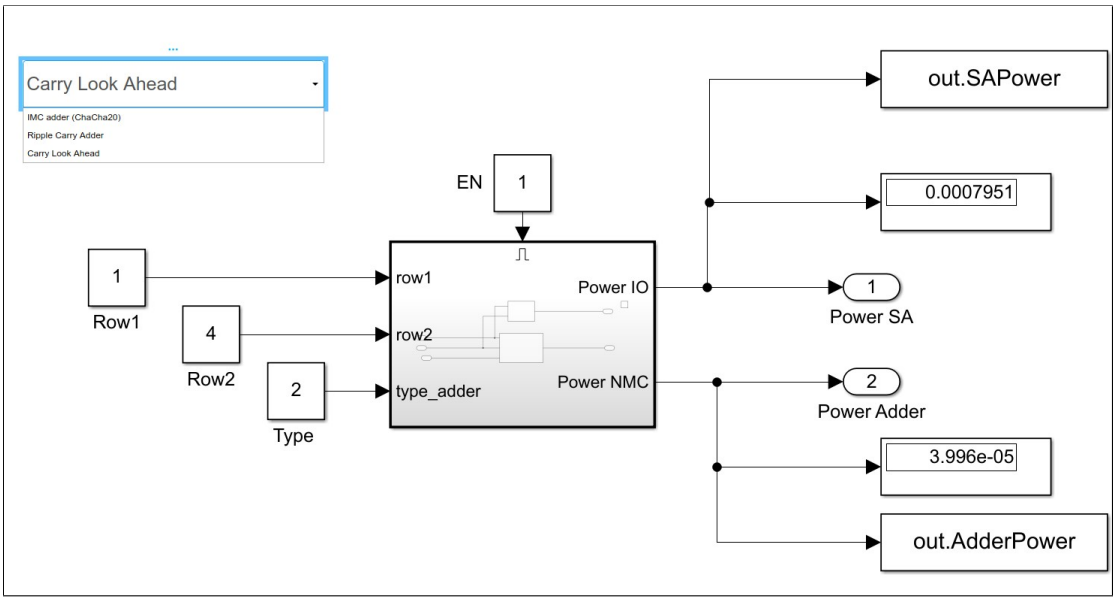


Figure 5.13: Overview: IO Block

5.5.1 Sense Amplifier (SA) Block

SA block is used to model the power consumption of a sense amplifier. It estimates the power consumption based on the offset values given to the sense amplifier. Various sense amplifiers are modeled based on the look-up table format of power estimation. In the library files, three different estimations of the sense amplifier are present.

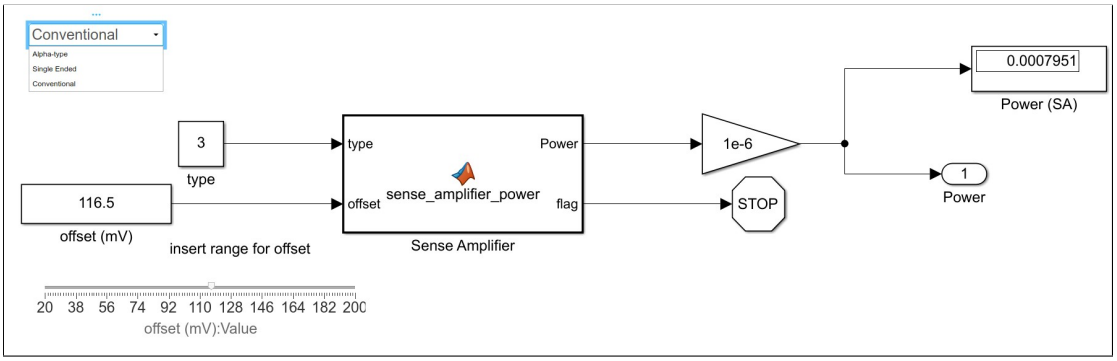


Figure 5.14: Sense Amplifier

5.5.2 Adder Block

Considering the NMC architecture of the ChaCha20-in-memory circuit proposed in [1], for emulating an arithmetic NMC block, an IMC-based adder circuit from [1] is added to the simulator. Since this adder takes IMC AND and IMC NOR output of bitlines, an IMC-operation block is added to calculate input for the adder based on data stored in the bitcell matrix. This block is modeled based on LUT templates and estimates power consumption based on current and previous inputs. Along with IMC-adder [1], models for carry-look ahead adder and ripple carry adder is also added Fig.-5.13.

5.5.3 Plot Data Block

After simulating the bitcell array, the simulator collects all the data related to bitline potential, cell currents, leakages, and power associated with each column. The plot data block then uses this data to generate graphs.

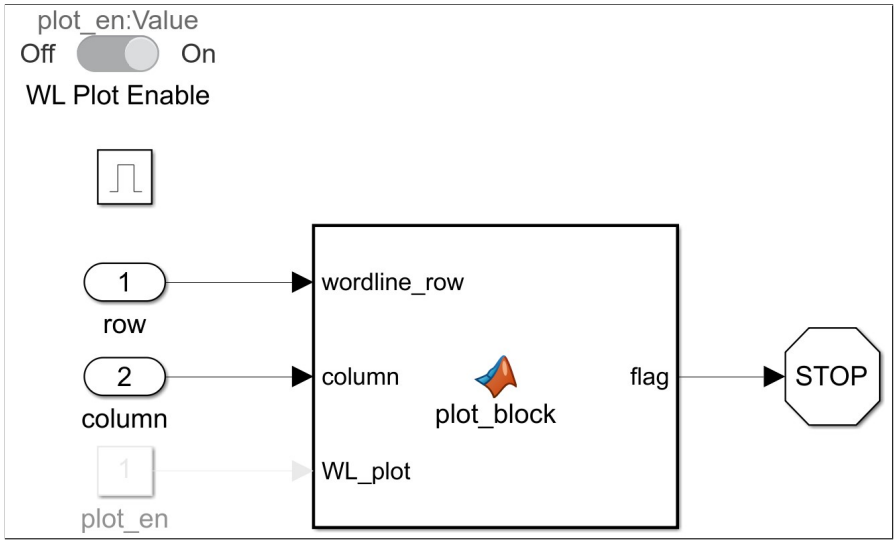


Figure 5.15: Plot Data Block

5.6 Library Files

The simulator needs pre-calculated power consumption data of the circuits, parameter values to create a memory array, and a few input-output files generated by the simulator for inter-block interactions. All these files and hierarchy are explained in this section.

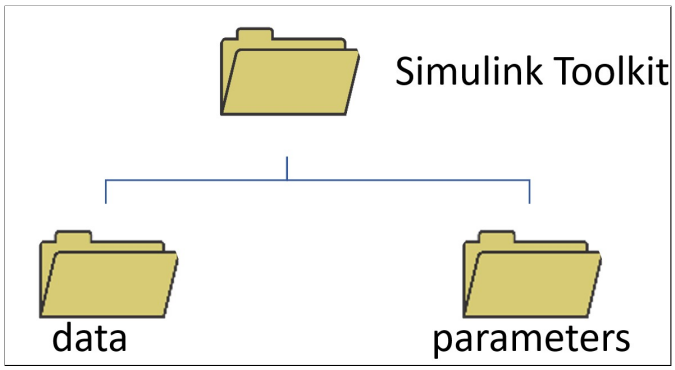


Figure 5.16: Overview: Library Files

Files Hierarchy overview

Broadly, the library file system is classified into two blocks. The first contains all the power consumption data and other output data generated by the simulator. The second block, i.e., the parameter block, contains the quantitative details to characterize the sub-circuits of the memory array.

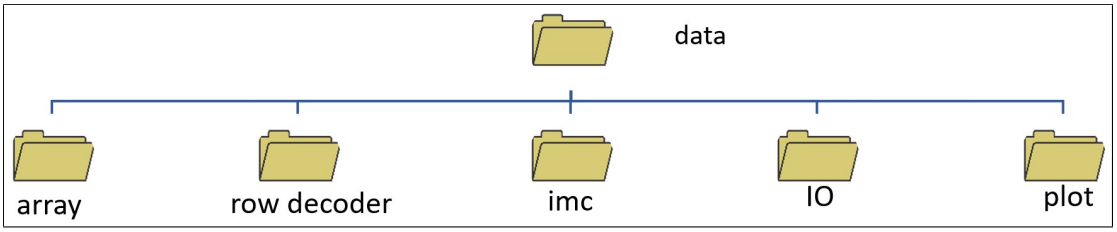


Figure 5.17: Data block

Parameter block contains the following files

- Array parameters
  It contains array structure information like the number of rows and columns, Bitline capacitance, Supply voltage, WL minimum pulse width, etc.
- Bitcell parameters
  It contains information to model a particular type of bitcell. It contains parameters like cell ratio, Threshold potential of NMOS and PMOS, different model parameters of MOS, type of bitline architecture, etc.
- Pre-Decoder Parameters
  This file collects information about the number and type of pre-decoder architecture simulated.
- Post Decoder Parameters
  This file collects information about the simulated post-decoder.

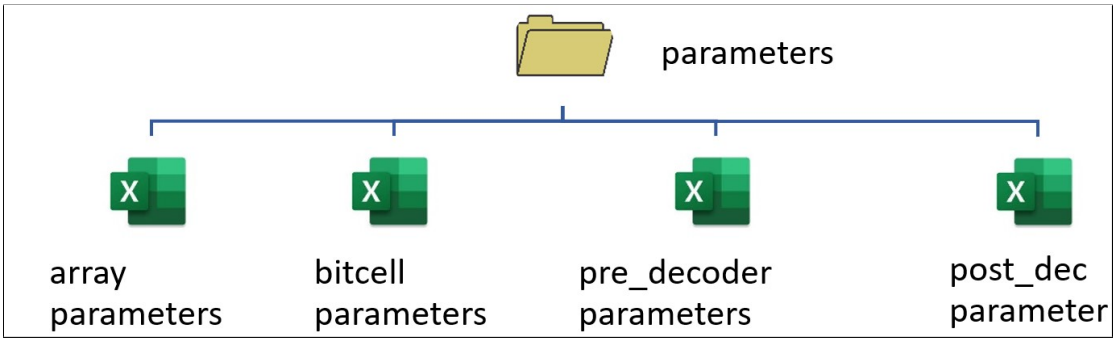


Figure 5.18: Parameter block

Array block contains the following two files

- Array
  This file contains the data matrix generated by the data generator to represent the bitcell memory array.
- WL
  This file contains the logical value of each word line generated by the data-generator block.

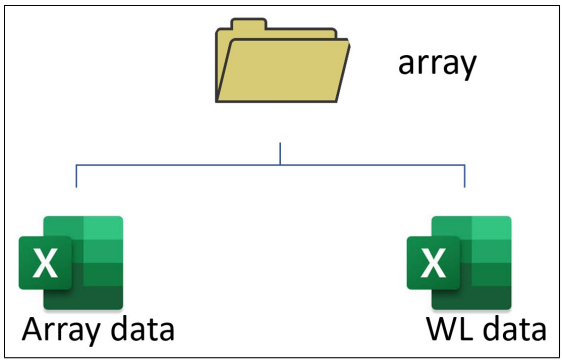


Figure 5.19: Array Block

Row decoder block contains the following two files

- Pre-Decoder input
  This file contains the binary converted data of the input row address. This is used to navigate LUT in the pre-decoder block.
- Pre-Decoder output
  This file contains the output of each pre-decoder sub-units (2x4,3x8, and 4x16) in decimal form.
- Power Consumption (Pre-Decoder)
  It contains power estimation LUT generated for different pre-decoder sub-units. These files are used to estimate power consumption.
- Post Decoder input
  Using the pre-decoder output file, input values for each post-decoder sub-unit (NAND, NOR) are generated and stored in this file. The post-decoder block further uses this file to navigate the power estimation LUT.
- Power Consumption (Post-Decoder)
  It contains power estimation LUT generated for different post-decoder sub-units. These files are used to estimate power consumption.

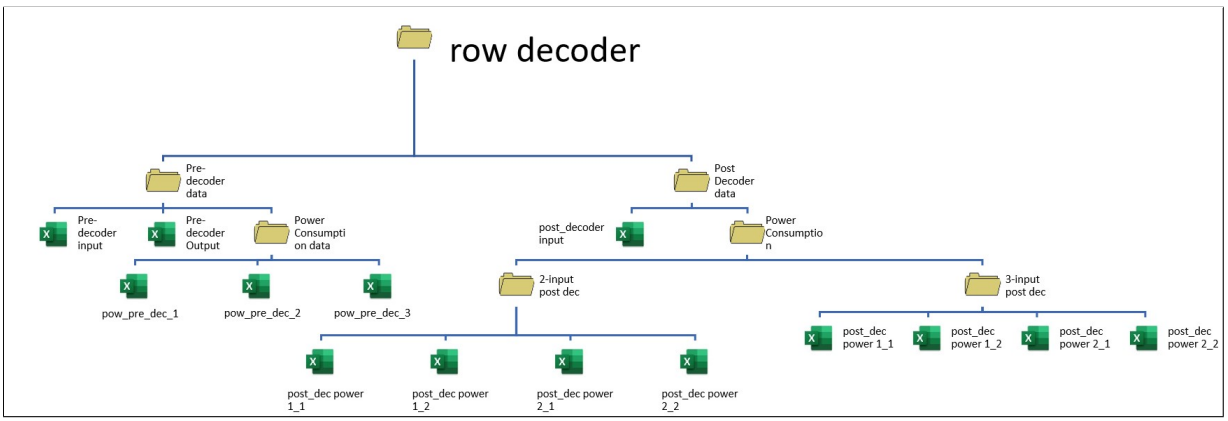


Figure 5.20: Row Decoder Block

IMC block contains the following two files

- IMC output It contains the logical values for bitline-IMC NOR and AND operations executed on selected rows of bitcell array.
- IMC previous output It contains the previous values for IMC output.

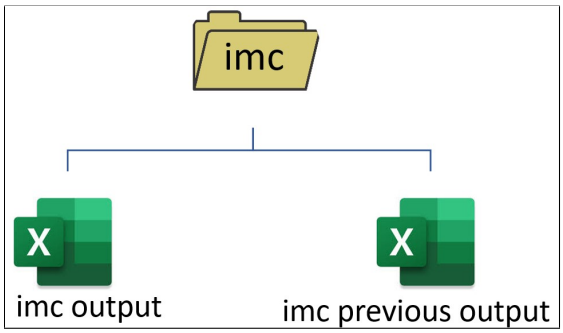


Figure 5.21: IMC block

IO block contains the following two sections

- Adder
  Contains the power consumption estimation LUT for Ripple carry adder, Carry select adder, and IMC adder.
- Sense Amplifier
  It contains the power consumption estimation LUT for alpha type, conventional, and single-ended sense amplifiers.

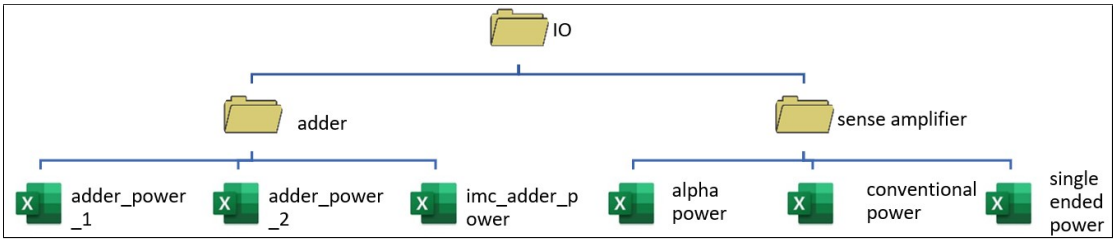


Figure 5.22: Overview: Library Files

Plot block contains the following files

- Bitline voltage
  These files record the simulated values of bitline potentials at different time instances for different columns.
- Cell Current
  These files record cell current values for each column simulated at different time instances for different columns.

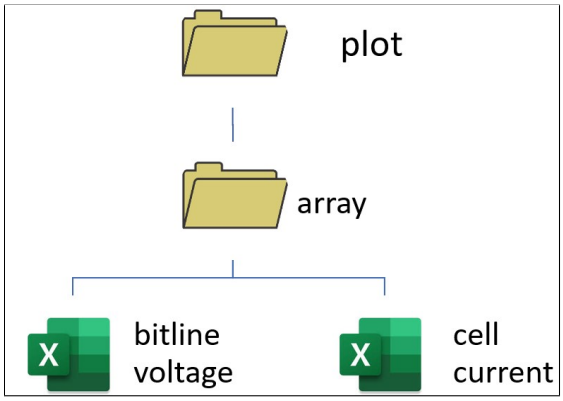


Figure 5.23: Overview: Library Files

CHAPTER 6

Results

6.1 Bitcell Array

6.1.1 Number of Bits in Word-line Pulse

Word lines are used in MAC operations supporting IMC arrays to send data (one of the operands) to the array. Data is sent on the word-line by converting it from numeric values to pulses of different widths according to the data value. Since the amount of bitline discharge depends on pulse width, based on the amount of bitline discharge, different values sent on the word-line can be distinguished. The method of conversion is explained below.

Consider a case where the designer needs to allot 3 bits to each word-line so that values ranging from 0 to 7 can be taken as feature data. So, value five can be represented in 3 bits as "101" in binary. The minimum WL pulse width is "dt" seconds. So, based on binary weightage, the pulse width will be calculated.

WL pulse width $= 8.dt.(\text{“1”}) + 4.dt.(\text{“0”}) + 1.dt.(\text{“1”}) = 9dt.$

Bitcell Array Power Consumption	
Number of Bits per word-line	Power Consumption (W)
1	0.000214889
2	0.000260283
3	0.000295871
4	0.000346903
5	0.000427352

Table 6.1: Number of bits vs. Power Consumption

The assumptions made for the estimation in Tab. 6.19 are:

- Array Size: 64x64 Matrix
- Bitcell: 6T cell
- Number of asserted word-line: 1
- Word-line data type: Decimal
- Minimum word-line pulse width: 100ps
- word-line on time: 3ns

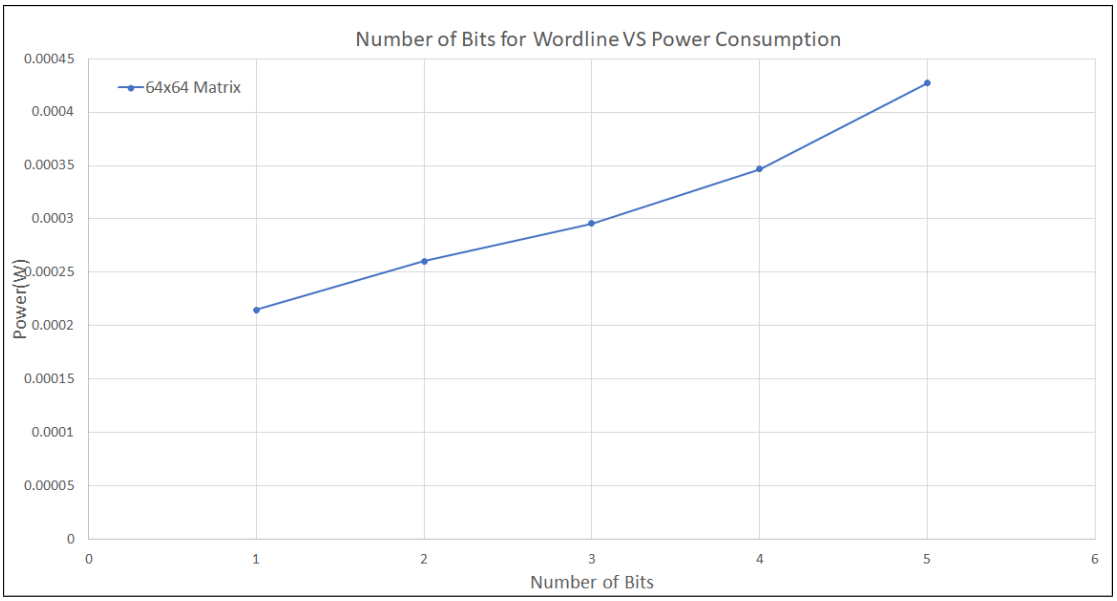


Figure 6.1: Number of bits for each word-line vs. Power Consumption

Observation

From the given Tab. 6.19 and plot in Fig. 6.1, we can observe that for higher word-line (WL) pulse widths, power consumption is higher because longer WL ON duration will cause more significant discharges in bitcells. So, the pre-charge circuit will draw more Power from the supply to recharge the bitlines.

6.1.2 Number of active Word-lines

In IMC operations, the WL of multiple rows is asserted at the same time. This causes multiple cells in a single column to contribute to cell current simultaneously. Since WL ON duration is fixed, higher cell current will cause higher bitline discharge, consequently leading to higher power consumption.

The assumptions made for estimations in Tab. 6.2 are mentioned below:

- Array Size: 64x64 Matrix
- Bitcell: 6T cell
- Number of bits for each word-line: 1
- word-line data type: Binary
- Minimum word-line pulse width: 10ps

Bitcell Array Power Consumption	
Number of ON WL	**Power Consumption (W)**
2	0.030596939
4	0.053365545
6	0.070327565
8	0.083679258
10	0.093639486
12	0.101860356
14	0.10865036
16	0.114117567
18	0.11854832
20	0.122233462
22	0.125710129

Table 6.2: Number of asserted word-lines vs. Power Consumption

Observation

From the given Tab. 6.2 and plot in Fig. 6.2, we observe that on activating more bitcells per column, the power consumption of the array increases. As more bitcells contribute to bitline discharge with the WL on duration remaining constant, bitline will discharge more for more number of activated word-lines. Hence, high work has to be done by pre-charge circuit to pre-condition the bitlines back to Vdd, again leading to higher power consumption.

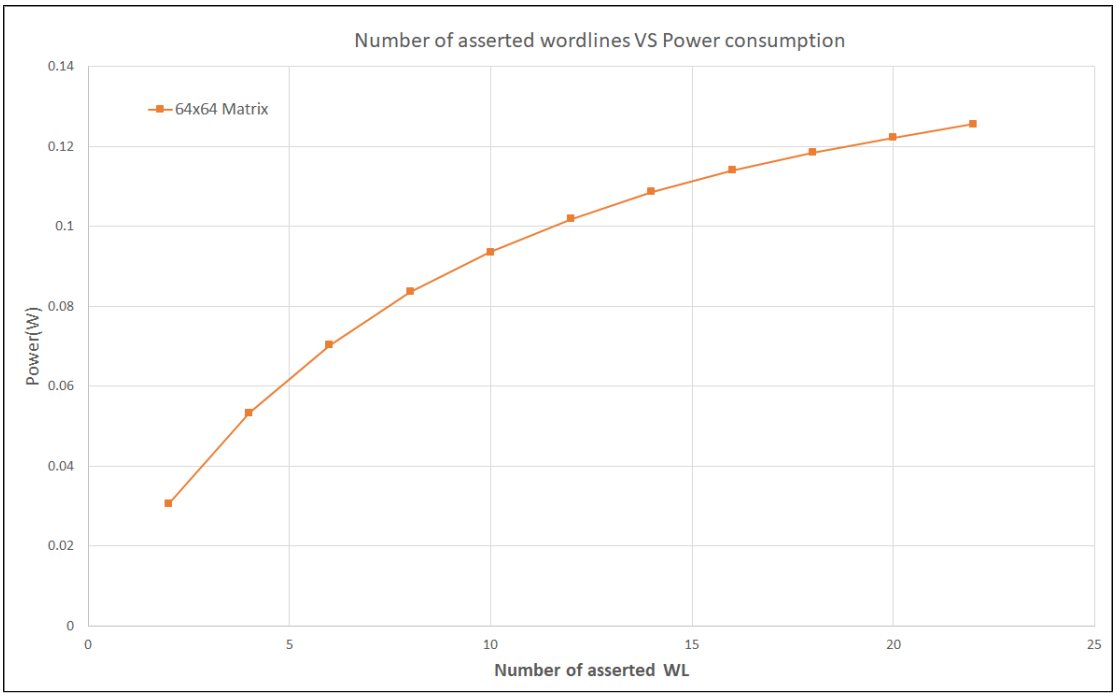


Figure 6.2: Number of asserted word-lines vs. Power

6.1.3 Effects of lowering word-line Voltages and power supply

Lowering (or boosting) the word-line (WL) and bitline (BL) voltages are some of the read and write assist schemes for bitcells. In these schemes, the WL or BL voltages are reduced (or boosted) from the usual Vdd.

Power supply lowering is a method used to reduce the power consumption of any circuit. In this, we reduce the power supply from its maximum value to some lower value, on which the circuit functionality can sustain. The following results depict the effect of word-line lowering and power supply lowering on power consumption.

The assumptions for the estimations in Fig. 6.3 are mentioned below:

- Array Size: 64x64 Matrix
- Bitcell: 6T cell
- Number of bits for each word-line: 1
- Number of asserted word-line: 1
- word-line data type: Binary
- Minimum word-line pulse width: 100ps

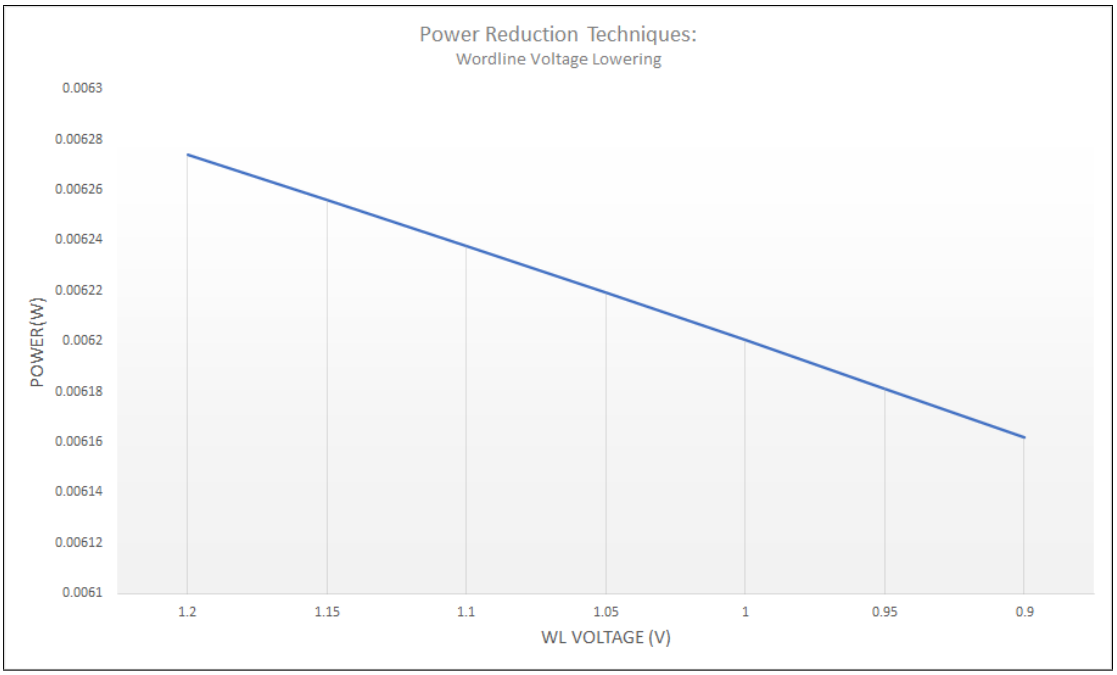


Figure 6.3: WL voltage vs. Power Consumption

Bitcell Array Power Consumption	
WL Voltage	**Power(W)**
1.2	0.006273982
1.15	0.006256104
1.1	0.006237909
1.05	0.006219386
1	0.006200527
0.95	0.006181324
0.9	0.006161767

Table 6.3: Number of asserted word-lines vs. Power Consumption

Lowering Supply Voltages

In the analysis, the supply voltage of bitcells, word-line voltages, and bitlines voltages are lowered and shown in Table 6.4 and 6.3.

Bitcell Array Power Consumption	
WL Voltage	**Power(W)**
1.2	0.006273982
1.15	0.006256104
1.1	0.006237909
1.05	0.006219386
1	0.006200527
0.95	0.006181324
0.9	0.006161767

Table 6.4: Vdd voltage vs. Power Consumption

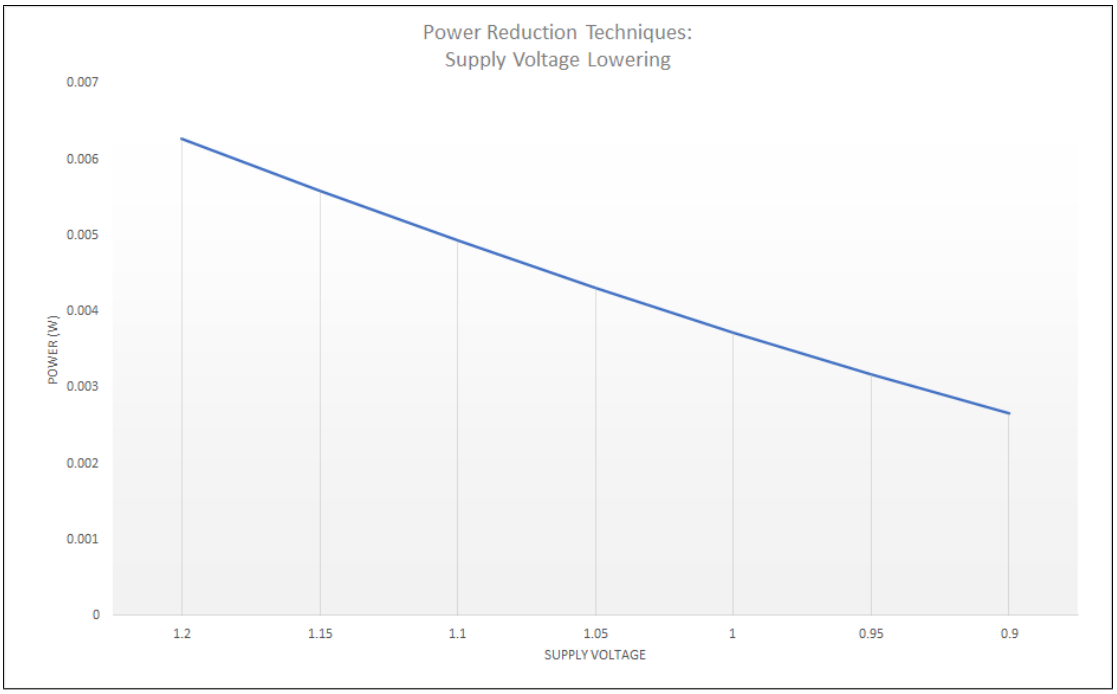


Figure 6.4: Supply voltage vs. Power Consumption

Observation

From the given Tab. 6.3, 6.4 and Fig. 6.3,6.4, it is observed that on lowering WL voltages and supply voltages, there occurs a steady decrease in power consumption of the bitcell array.

6.1.4 Power based Side Channel Attack Analysis

Power-based side-channel attacks are the class of indirect attacks to extract information about the stored/processed confidential data. These attacks depend on the correlation between the power consumption of the system and the data being processed. If the power-consumption patterns change drastically with the data being processed in a circuit, that circuit is classified as highly vulnerable.

In the analysis, we simulated read operation for three different classes of bitcells, namely, single-ended bitline (8T), differential bitline pair (6T), and complementary single-ended bitline pairs (10T). Here, we altered the data matrix in each simulation. Data matrix **"A"** represents checkerboard patterns, whereas data matrices from **"B through F"** represents 1's-matrix with the percentage of 1's varying from 10-50%. The graphs in Fig. 6.5 show a comparison of power consumption with varying data matrices.

Assumptions :

- Array Size: 64x64 Matrix
- Bitcell: 6T cell
- Number of bits for each word-line: 1
- Number of asserted word-line: 1
- word-line data type: Binary
- Minimum word-line pulse width: 100ps

Bitcell Array Power Consumption for different Bitcells (W)					
Bitcell Data-Matrix Type			Power Consumption		
			6T	8T	10T
1	**Checker Board**	A	0.006207019	0.003842674	0.005831
2	**1- Matrix : 10**	B	0.005755189	0.000923516	0.00578
3	**1- Matrix : 20**	C	0.005907341	0.002360431	0.005788
4	**1- Matrix : 30**	D	0.006106478	0.002393173	0.005813
5	**1- Matrix : 40**	E	0.006139387	0.003521833	0.005826
6	**1- Matrix : 50**	F	0.005321539	0.004289945	0.005836

Table 6.5: Power consumption for different bitcells under different data

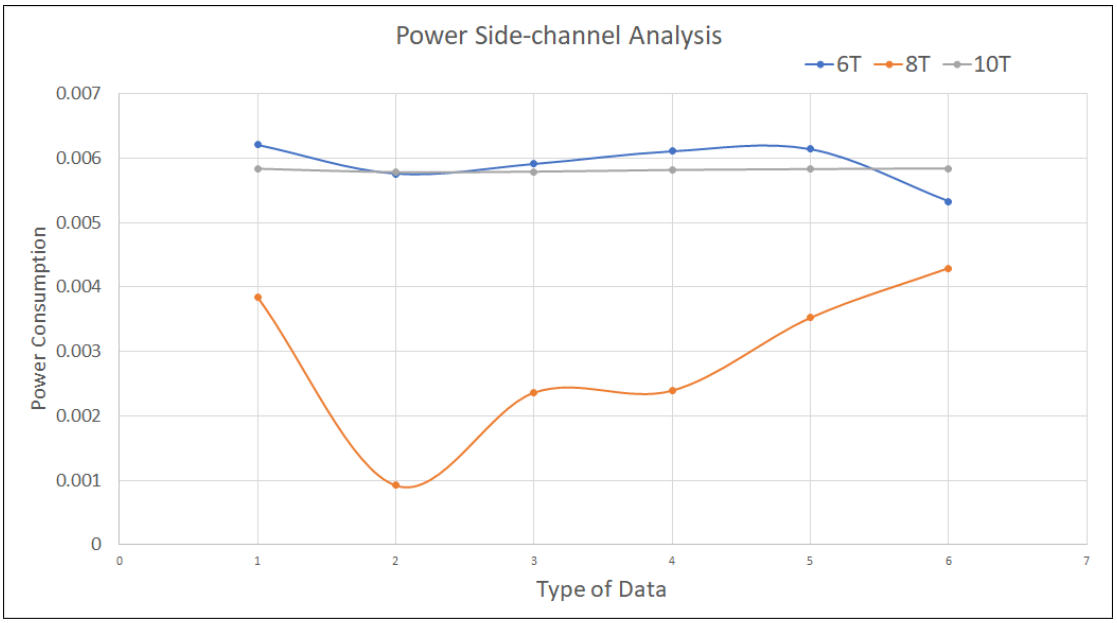


Figure 6.5: Power Consumption vs. Data Matrix

Observation

From the given Tab. 6.5 and Fig. 6.5, we can observe that out of the three bitcells, the single-ended bitline bitcell (8T bitcell) has the maximum variation in power consumption patterns when the data matrix is altered. The other bitcells, 6T and 10T, are either stable (complementary single-ended bitline pairs (10T)) or show little variations in power consumption with varying data. So, single-ended bitline family bitcells are highly vulnerable to power-based side-channel attacks compared to the other two groups.

6.1.5 Size of Bitcell Array

In this analysis, the effect of increasing the size of the bitcell array on power consumption is studied. Bitcells in rows where a word-line is asserted is called active cells. They are the primary and targeted contributors to cell current. However, there are other cells in the rows where the word-line is not asserted. The cells of these rows are not activated and ideally should not contribute to cell current. Nevertheless, due to effects like sub-threshold conduction, these cells also contribute to the cell current in the form of leakages. Secondly, more bitcells per column would result in a higher bitline capacitance, and a larger capacitor will require more power to charge.

Above these two facts are two of the many factors which cause an increase in power consumption with the size of the bitcell array.

Array Size	**Power (W)**
16x16	0.000532154
32x32	0.001800333
64x64	0.006273982
128x128	0.019329237
256x256	0.052537186
512x512	0.129769197
1024x1024	0.355914819

Table 6.6: Power consumption for different sized bitcell-array

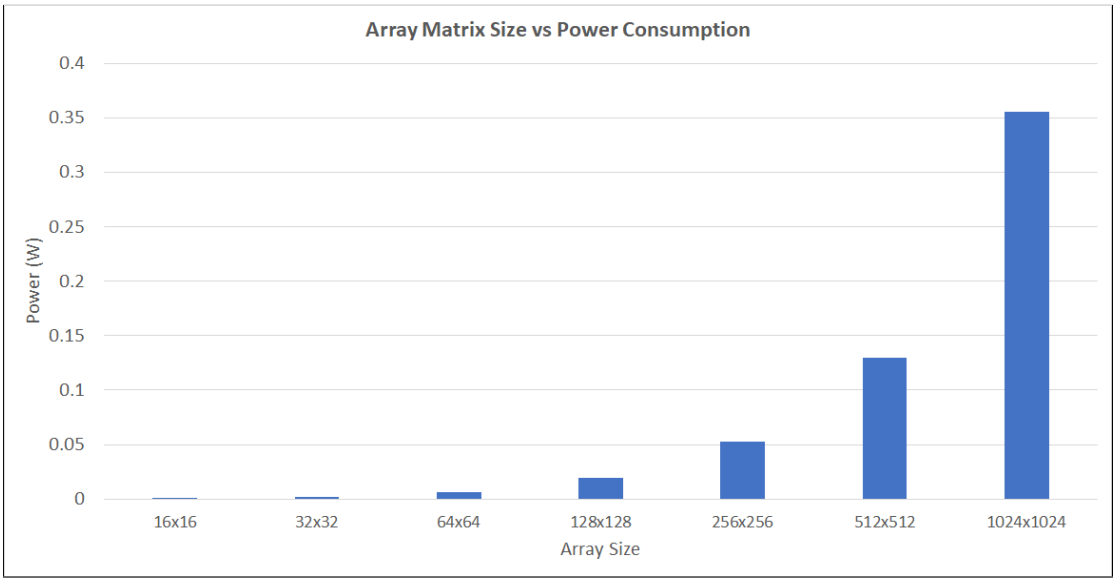


Figure 6.6: Power Consumption vs. Size of bitcell array

6.2 Row Decoders

6.2.1 Pre-Decoders

Case1: Comparing Power consumption of different combinations of pre-decoders. Pre-decoders are the first stage of the row-decoders block. The user must select 2x4, 3x8, and 4x16 pre-decoders in the circuit based on the required number of input address lines. In the following analysis, power consumption for different such configuration of pre-decoder is simulated.

Number of Input row address lines	**Row Address**	**2x4**	**3x8**	**4x16**	
		Count	**Count**	**Count**	**Power (uW)**
0	0	0	0	0	0
4	15	2	0	0	35.96
5	31	1	1	0	56.2
6	63	1	0	1	71.8
6	63	3	0	0	53.94
7	127	2	1	0	74.1
8	255	1	2	0	94.42
8	255	2	0	1	84.88
9	511	1	1	1	90.95
10	1023	1	0	2	125.6

Table 6.7: Power Consumption vs. offset

Case2: Comparing two combinations of Pre-decoder for different input addresses. In the following analysis, two pre-decoder configurations for an equal number of input address lines are selected and simulated for different row addresses.

	2x4	3x8	4x16
Config 1	1	2	0
Config 2	2	0	1

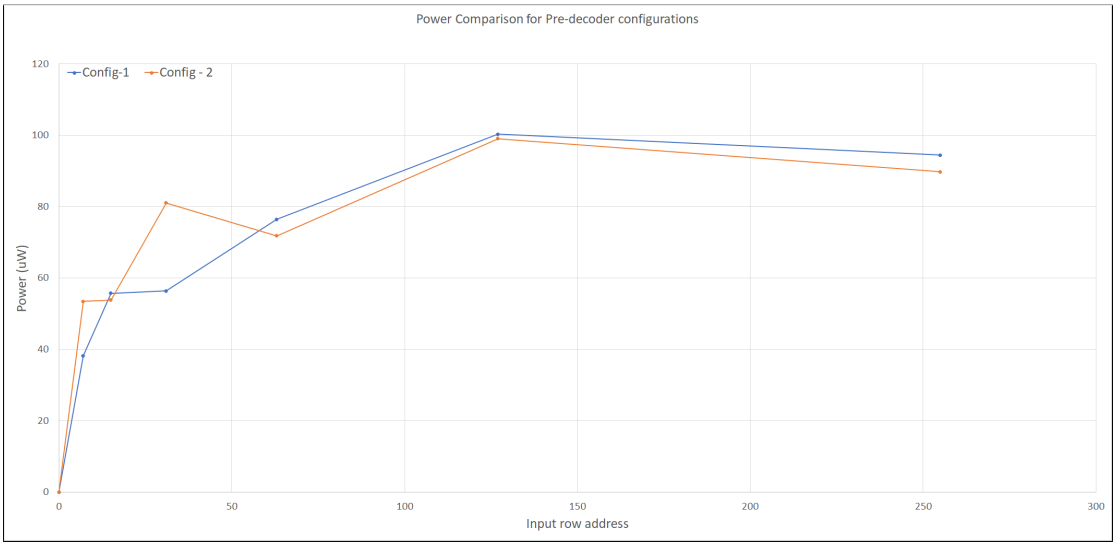


Figure 6.7: Power Comparison for Pre-decoder configurations

Input row Address	**Power (uW)**	
	Config-1	Config - 2
0	0	0
7	38.221	53.455
15	55.74	53.833
31	56.401	81.05
63	76.422	71.813
127	100.3	99.034
255	94.42	89.793

Table 6.8: Power comparison between two pre-decoder configurations for different input address

6.2.2 Post-Decoders

Based on the number of pre-decoder units, the number of post-decoder units, as well as the number of inputs of each post-decoder, is calculated.

Case1: Power consumption: Different type of 2-input Post Decoders

In this analysis, all the different pre-decoder configurations which result in 2-input post decoders configurations are simulated.

Pre-Decoder Configuration				
2x4	3x8	4x16	Number of input address lines	
Count	Count	Count		Number of WL
0	0	0	0	1
2	0	0	4	16
1	1	0	5	32
1	0	1	6	64

Table 6.9: Pre-decoder configurations for 2-input post-decoder

	2-input Post Decoder Power(uW)			
	NAND		NOR	
Number of WL	Static	Dynamic	Static	Dynamic
1	0	0	0	0
16	84.247	130.392	231.1	238.9
32	85.521	42.461	363.1	380.9
64	88.071	42.696	627.1	664.9

Table 6.10: Power comparison between two pre-decoder types for different number of wordlines

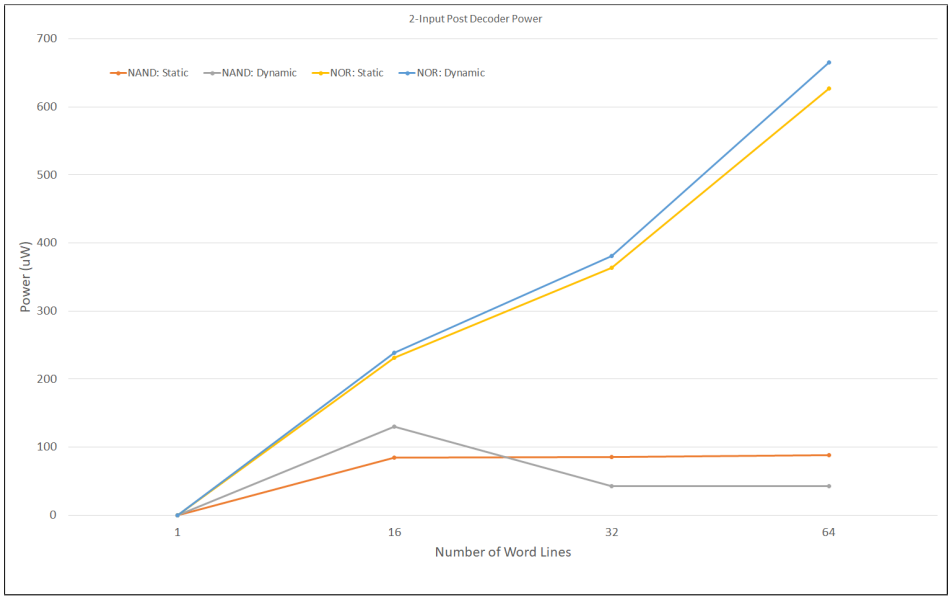


Figure 6.8: Power Consumption vs. offset

Case2: Power consumption: Different type of 3-input Post Decoders.

In this analysis, all the different pre-decoder configurations which result in 3-input post decoders configurations are simulated.

Pre-Decoder Configuration				
2x4	3x8	4x16	Number of Input	
Count	Count	Count	Address Lines	Number of WL
2	1	0	7	128
2	0	1	8	256
1	1	1	9	512
1	0	2	10	1024

Table 6.11: Pre-decoder configurations for 3-input post-decoder

	3-input Post Decoder Power(uW)			
	NAND		NOR	
Number of WL	Static	Dynamic	Static	Dynamic
128	45.869	44.032	1488.1	376.408
256	48.568	45.752	2728.1	568.67
512	55.818	47.173	4621.3	689.29
1024	70.317	50.015	8407.7	930.53

Table 6.12: Power comparison between two pre-decoder types for different number of word-lines

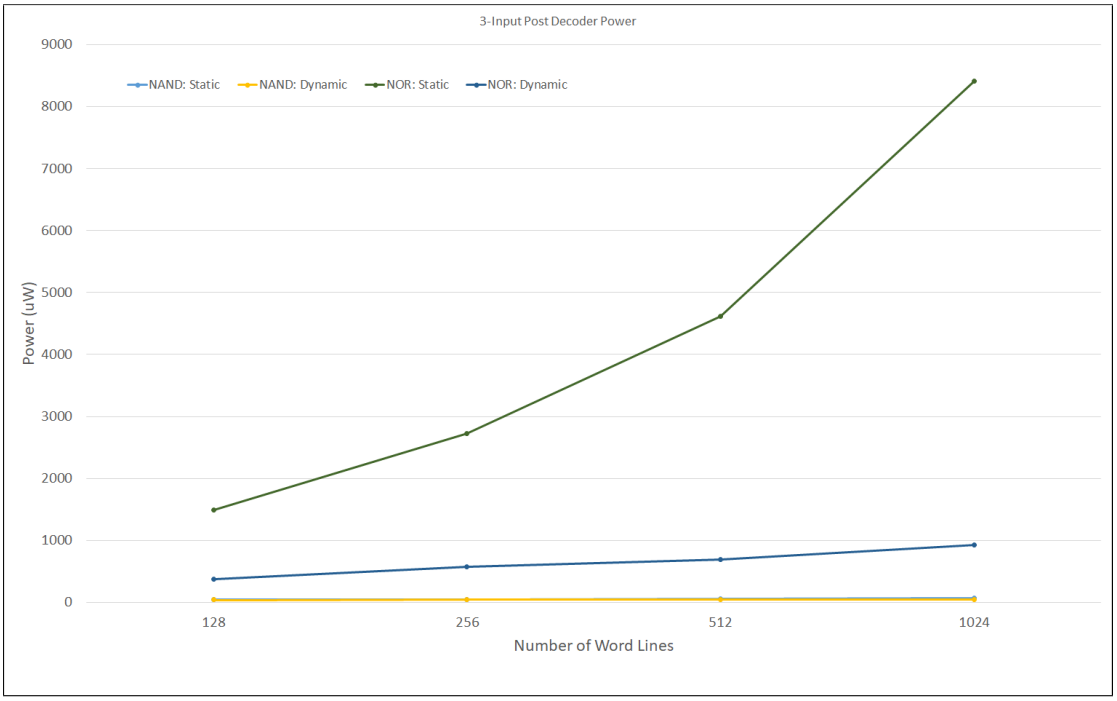


Figure 6.9: Power Consumption vs. offset

6.3 IO and Near Memory Compute block

6.3.1 Sense Amplifiers

Sense amplifiers require a minimum potential difference between the two input nodes to amplify that difference to logic-"0" and logic-"1" at the output.

Case 1: Power Consumption vs. provided offset

As we know, sense amplifiers require a minimum difference between the two nodes. In the following section, sense amplifiers are given an increasing potential difference across their terminal (offset) to analyze the effect on power consumption when the offset potential difference is given more than what the sense amplifier requires. Here, the power profile data for three different sense amplifiers is presented.

- **Conventional Sense Amplifier**

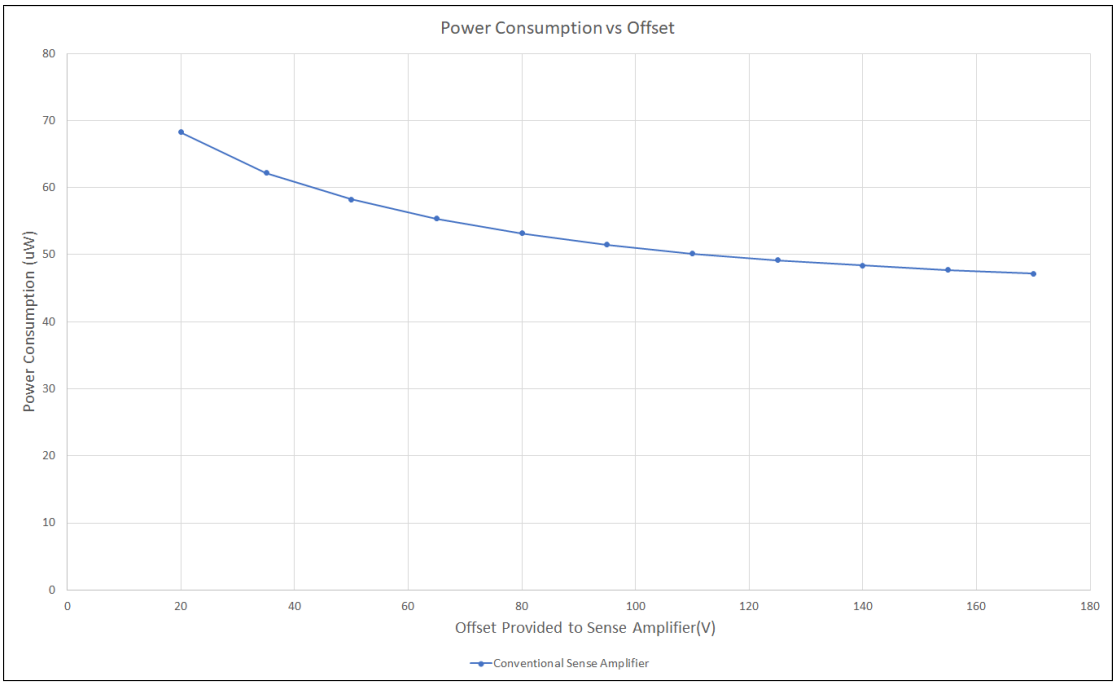


Figure 6.10: Power Consumption vs. offset (Conventional Sense Amplifier)

Offset Provided (mV)	Power Consumption (uW)
20	68.28
35	62.16
50	58.25
65	55.35
80	53.17
95	51.47
110	50.16
125	49.16
140	48.37
155	47.72
170	47.17

Table 6.13: Power Consumption vs. offset (Conventional Sense Amplifier)

- **Alpha-Type Sense Amplifier**

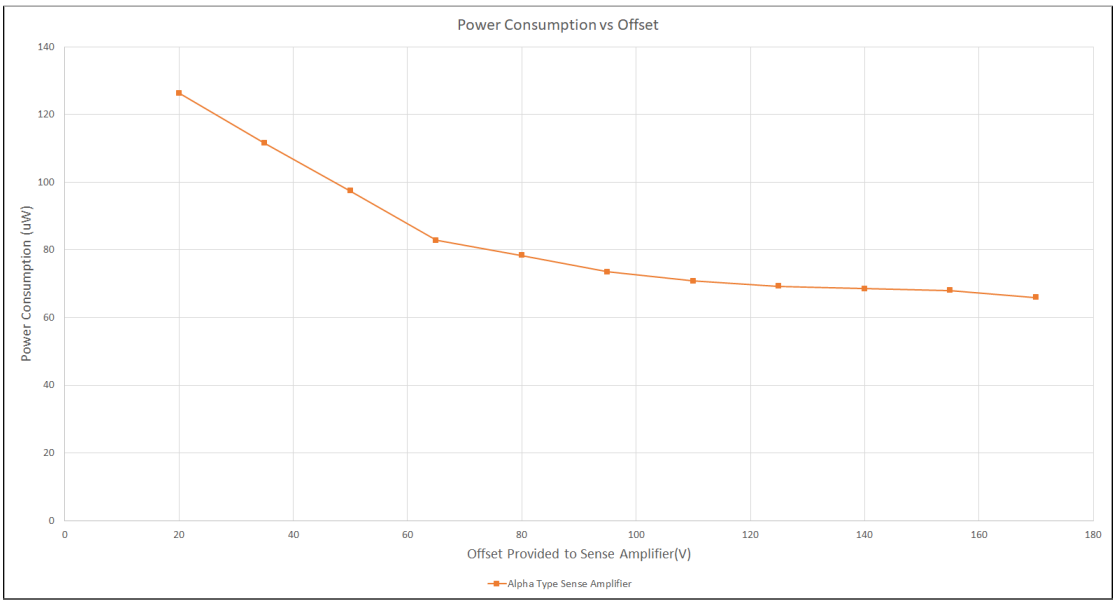


Figure 6.11: Power Consumption vs. offset (Alpha-Type Sense Amplifier)

Offset Provided (mV)	Power Consumption (uW)
20	126.3
35	111.5
50	97.4
65	82.83
80	78.4
95	73.5
110	70.8
125	69.3
140	68.5
155	68
170	65.98

Table 6.14: Power Consumption vs. offset(Alpha-Type Sense Amplifier)

- **Single Ended Sense Amplifier**

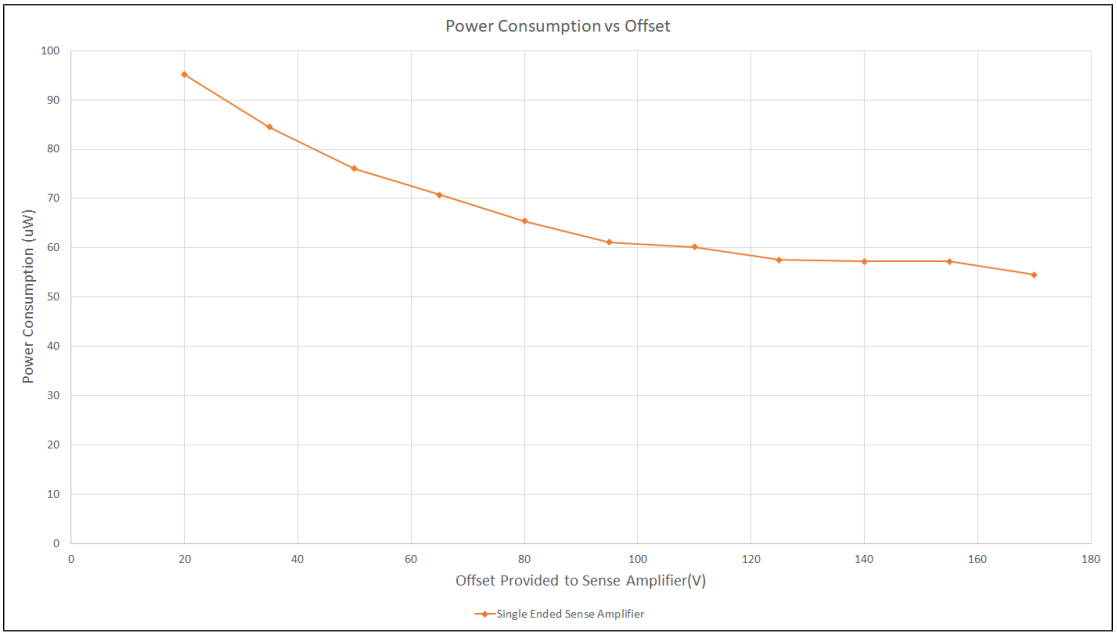


Figure 6.12: Power Consumption vs. offset (Single Ended Sense Amplifier)

Offset Provided (mV)	Power Consumption (uW)
20	95.14
35	84.52
50	76.06
65	70.73
80	65.39
95	61.09
110	60.18
125	57.59
140	57.26
155	57.23
170	54.52

Table 6.15: Power Consumption vs. offset (Single Ended Sense Amplifier)

Observation

It is observed that on giving a higher potential difference across the sense amplifier input nodes (bitlines), the power consumption of all three sense amplifiers decreases.

Case 2: Power Comparison among sense amplifiers for different column width of bit-cell Array

Sense amplifiers are used at the output of bit lines to decode the slight discharge done by the bitcell of that column. If the number of columns in a bitcell array is increased, then the required number of sense-amplifier at IO would also increase. Hence, more units will cause an increase in power consumption.

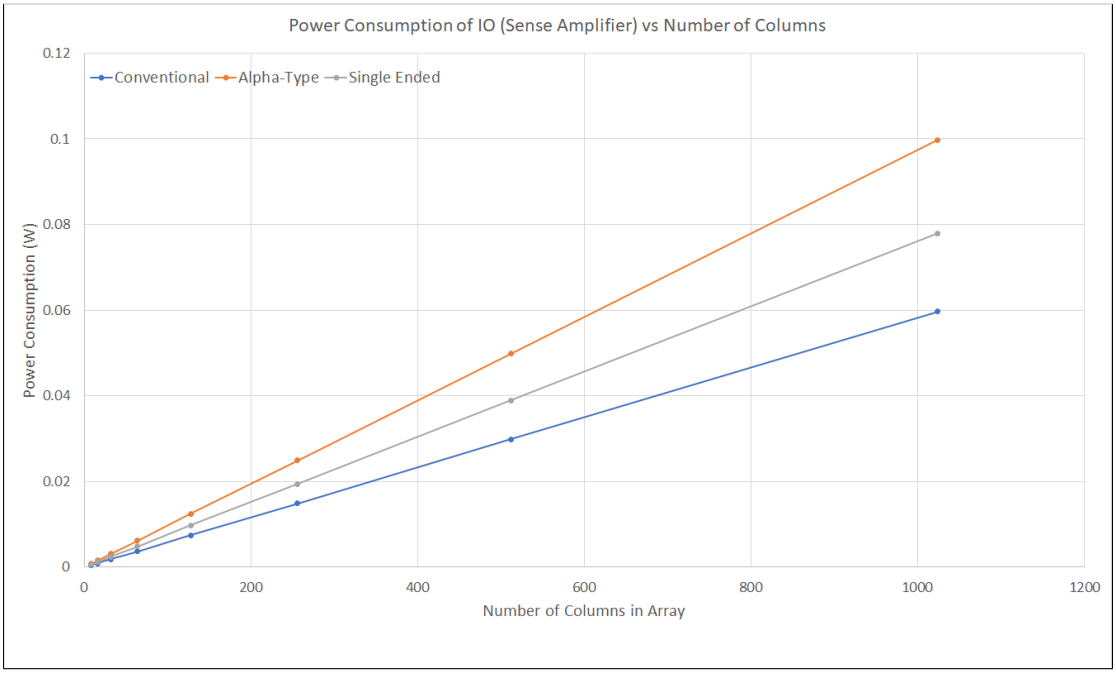


Figure 6.13: Power Comparison: Different Column size of Bitcell Array

Bitwidh of Bitwidh of element	**Power (W)**		
	Conventional	Alpha-Type	Single Ended
8	0.000466	0.000779	0.000608
16	0.000932	0.001558	0.001217
32	0.001864	0.003117	0.002434
64	0.003728	0.006234	0.004868
128	0.007456	0.012467	0.009736
256	0.014912	0.024934	0.019471
512	0.029824	0.049869	0.038943
1024	0.059648	0.099738	0.077885

Table 6.16: Power Comparison: Different Column size of Bitcell Array

6.3.2 NMC block : Adder

Power Comparison of Near Memory Adders

Data Stored in the array is read and computed directly, without the need to move them through the data bus. This is carried out using near-memory computing elements. Three NMC adders in the simulator database are simulated in the following analysis.

Case 1: Power variation with different input operands.

To check the dependency of power consumption on the input data, different rows of the bitcell array are passed to the adders. In the following results, the power consumption of each adder is recorded for different array rows (input operand).

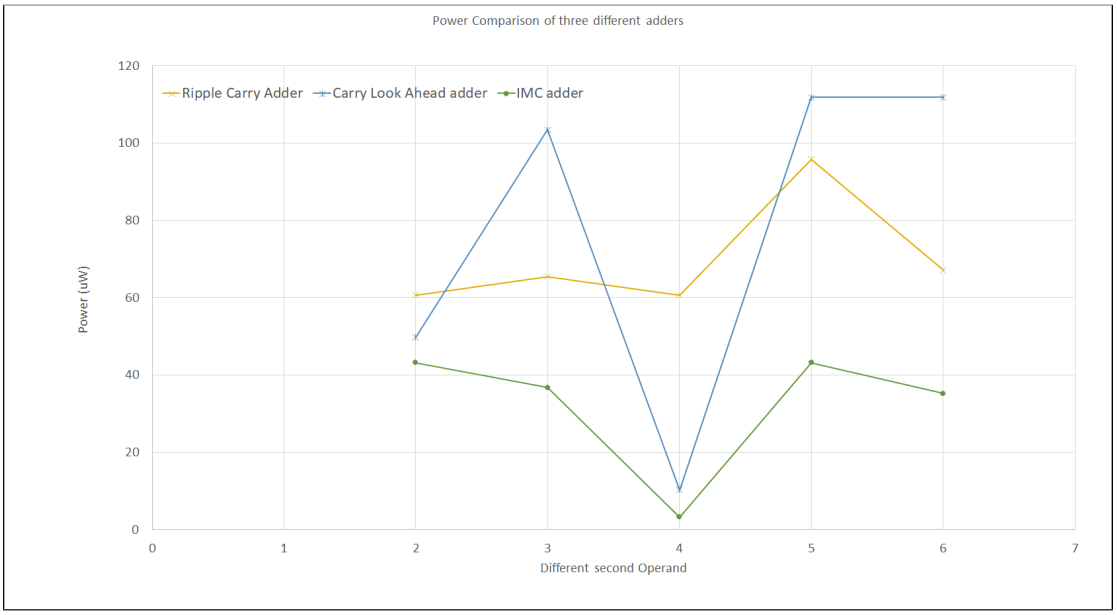


Figure 6.14: Power Consumption vs. Different Data

Second Row	Power (uW)		
	Ripple Carry Adder	**Carry Look Ahead Adder**	**IMC adder**
2	60.73701	49.8271	43.19199
3	65.52201	103.4204	36.81198
4	60.73701	10.3373	3.316939
5	95.82705	111.8825	43.19199
6	67.11702	111.8825	35.21698

Table 6.17: Power Comparison : Different data elements

Case 2: Power variation with different sizes of operands (altering size of bitcell matrix).

In these results, the bit-width of input operands to adders is increased. This is done by increasing the size of the underlying bitcell array matrix.

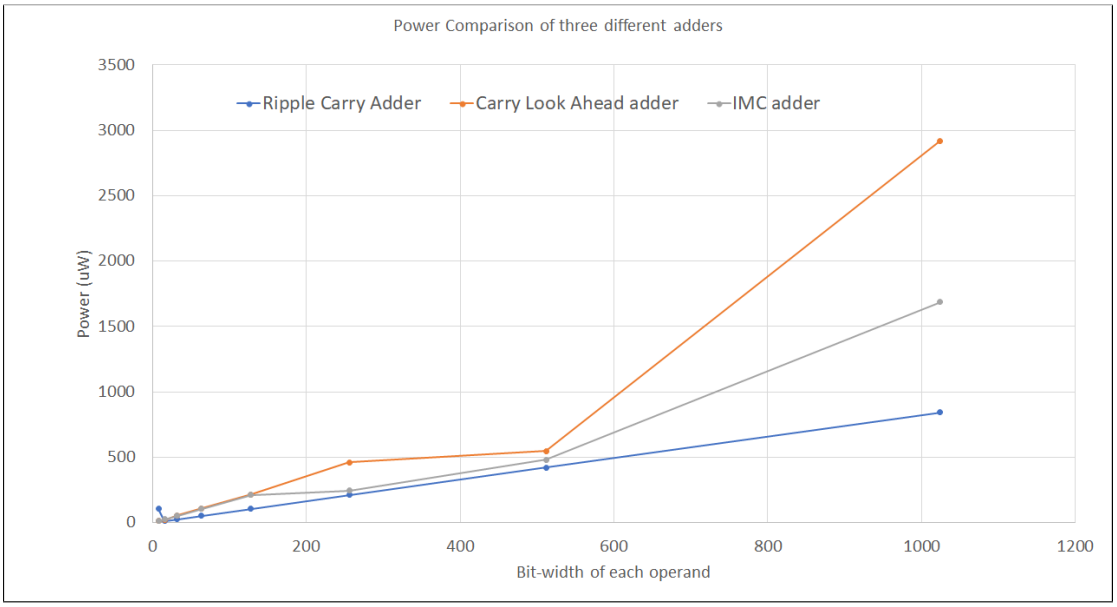


Figure 6.15: Power Consumption vs. Different Data

Bitwidh of elements	Power (uW)		
	Ripple Carry Adder	Carry Look Ahead Adder	IMC adder
8	105.3970633	14.3379	13.17463
16	13.17463291	20.2137	26.34927
32	26.34926583	54.5309	52.69853
64	52.69853166	109.0618	105.3971
128	105.3970633	215.3029	210.7941
256	210.7941266	461.6335	246.138
512	421.5882533	546.2545	479.5161
1024	843.1765065	2918.4	1686.353

Table 6.18: Power Comparison : Different Bit-width of elements

Observation

On increasing the bit-width of each input operand, the adder block increases power consumption since more adder circuits will be required.

6.4 Test-Case Simulation

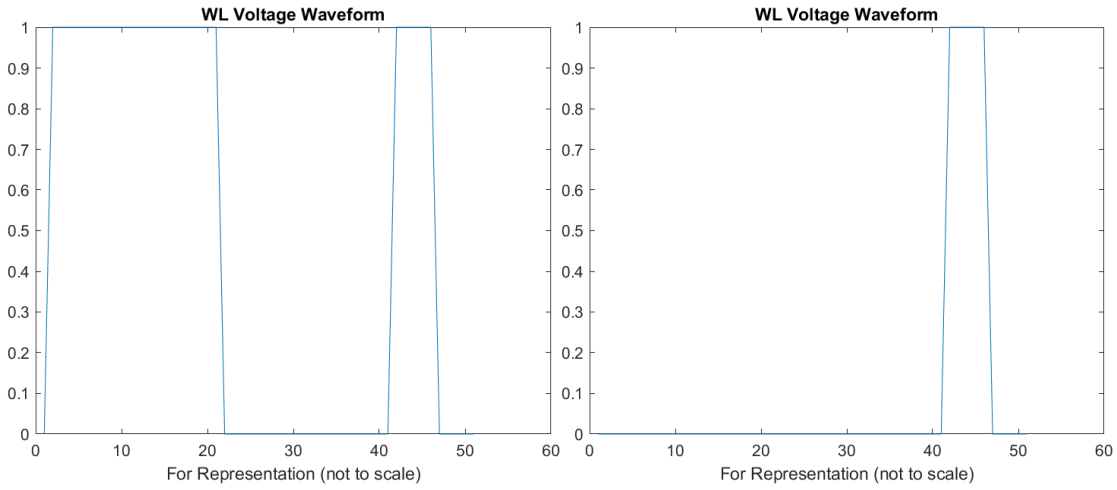
Parameters of the test-case IMC array:

- Array Size : 64x64.
- Since 64 wordlines(rows) are required to be generated, input address bit-width must be of 6-bits.
- For 6-bit input address line, three 2x4 pre-decoders are used.
- For Post-decoder, three input static NAND decoders are used.
- Two wordlines are activated with feature data value 5 and 1.
- Conventional Type Sense amplifier is used,
- For NMC block, IMC adder is used.

Output plots for the above test-case scenario is presented in the following results.

6.4.1 Wordline Plot

- Wordline
  - Two wordlines rows are activated.
  - Each wordline rows are given a bit-width of three bits.
  - Value sent on wordline row1 = 5 ("3'b101") and on wordline row2 = 1("3'b001").



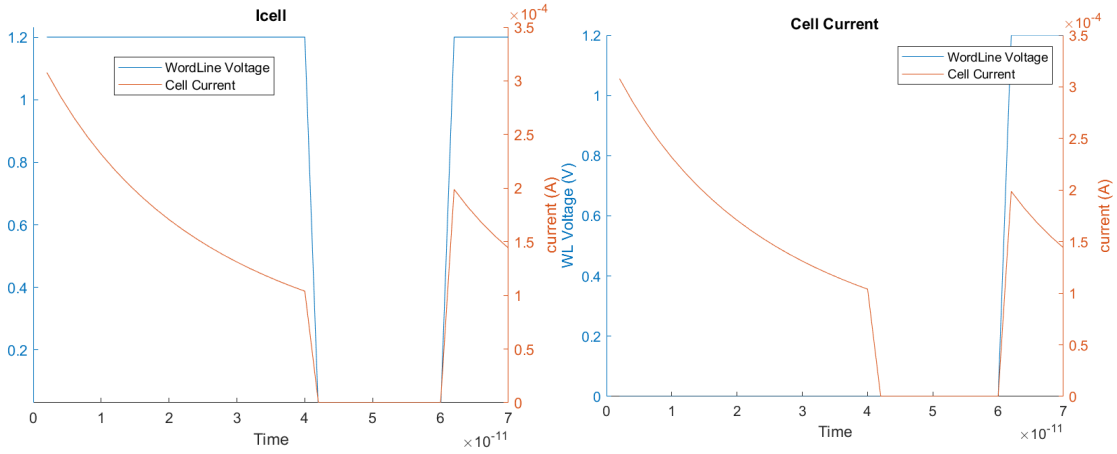
(a) Wordline pulse for "101"

(b) Wordline pulse "001"

Figure 6.16: Wordline Pulses for two different rows

6.4.2 Cell Current Plot

In the following graphs, the effect of multiple asserted wordlines on bitcell current is shown. In the results, the cell current of column-1 is shown along with the wordline voltages of row-1 and row-2. This is done to show the contribution of each activated wordline on the cell current of the same column. Since the wordlines of other rows are de-asserted(0), they are not shown in the graphs.



(a) I_{cell} for column: 1 with V_{WL} of row: 1

(b) I_{cell} for column: 1 with V_{WL} of row: 2

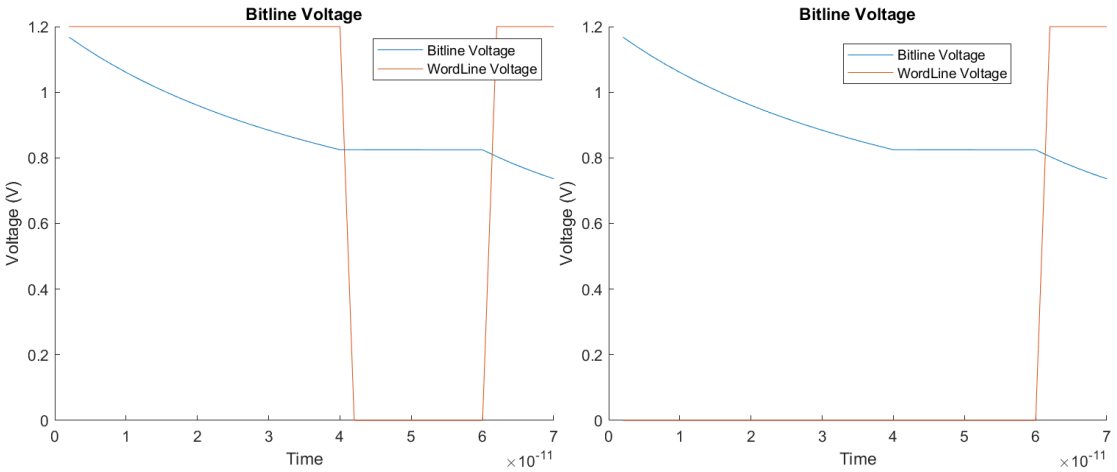
Figure 6.17: Cell Current and wordline potential for two different rows

Observation

- Wordline Row: 1
  Since three bits are assigned to each word line, the maximum pulse width it can accommodate is 70ps, as the minimum pulse width is 10ps. Here, for "101" pattern, wordline for row 1 is asserted for a total of 50ps (40+10) Fig.6.17a.
- Wordline Row:2
  Here, the wordline pulse pattern for "001" is represented. Based on the binary weight, wordline pulse width is 10ps Fig.6.17b.
- Cell Current corresponding to row:1
  It is observed that, at the assertion of WL for row-1, a cell current is developed. This current decays with time due to bitline discharge.
- Cell Current corresponding to row:2
  Here, we observe a sudden increase in cell current at 60ps. This is due to the assertion of the wordline for both row1 and row2 during the 60-70ps time interval.

6.4.3 Bitline Voltage Plot

As a result of cell current, there is a discharge in the potential stored on floating bitlines(BL). In the following graphs, the effect of multiple asserted wordlines on bitline discharge is shown.



(a) V_{BL} for column: 1 with V_{WL} of row: 1 (b) V_{BL} for column: 1 with V_{WL} of row: 2

Figure 6.18: Bitline Potential and wordline potential for two different rows

Observation

- In Fig6.18a and 6.18b, we see a steady decrease in bitline for initial 50ps, then a faster decay for last 10ps.
- This can be attributed to the fact that in first 50ps, only the wordline of row1 was asserted, while in the last 10ps, wordlines of both row1 and row2 were asserted.

6.4.4 Leakage Current Plot

In Fig. 6.19 bitline and bitline bar leakage currents for each column are shown. These leakage currents are due to inactive bitcells of the same column.

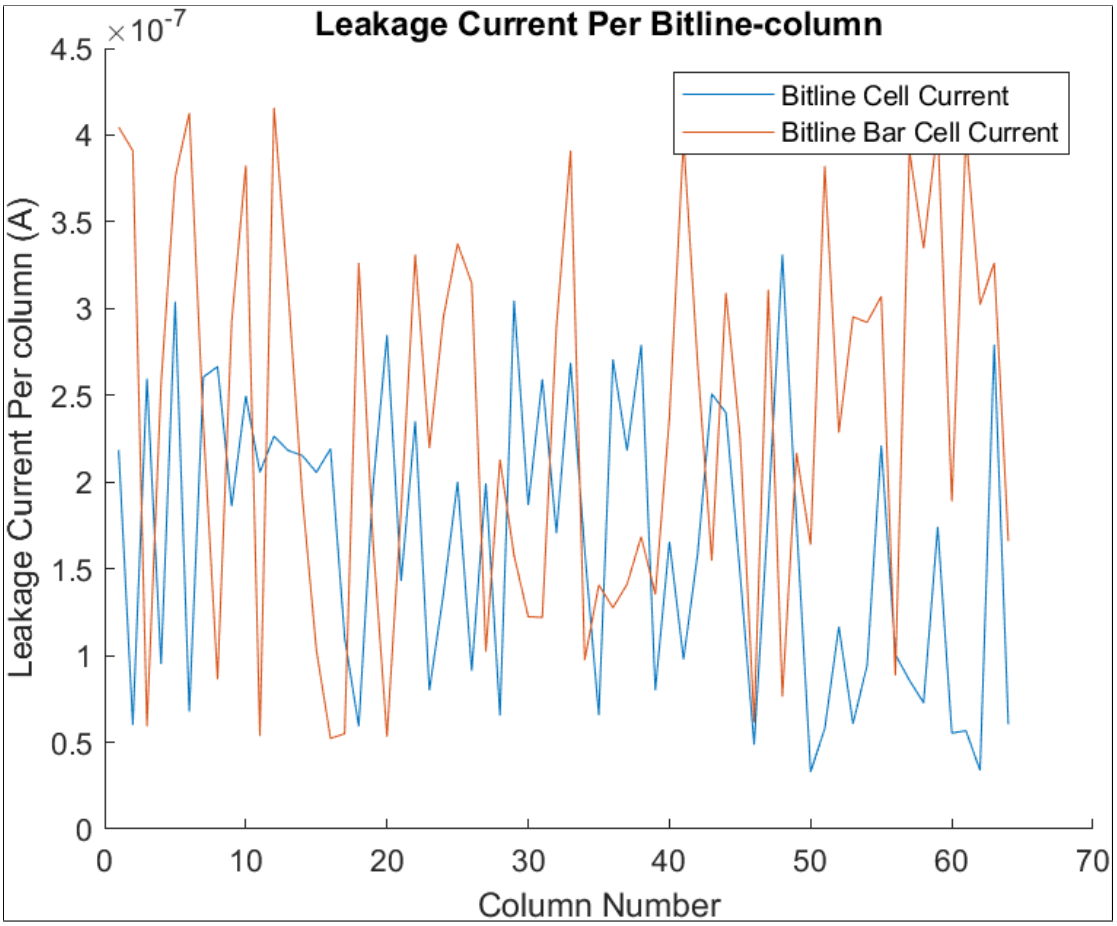


Figure 6.19: Leakage vs. Column

6.4.5 Power Consumption Per Column Plot

In Fig.6.20, each column's contribution to power consumption is shown.

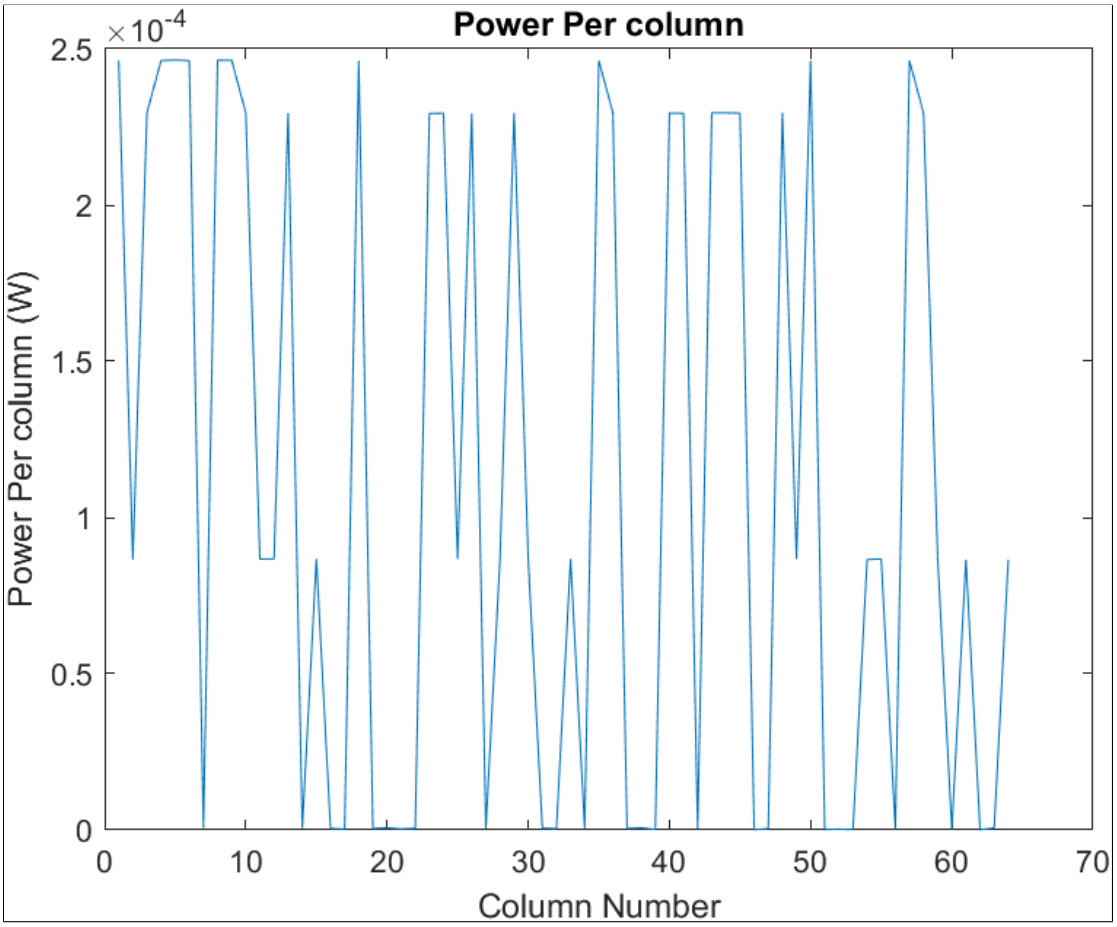


Figure 6.20: Power Consumption vs. Different Columns

6.4.6 Power Consumption Values

Generated Results

IMC Array Power Consumption	
Block	Power Consumption (W)
Pre-Decoder	27.22e-6
Post-Decoder	23.76e-6
Bitcell Matrix	0.00781
Sense Amplifier	0.003033
IMC adder	19.27e-06

Table 6.19: Number of bits vs. Power Consumption

REFERENCES

[1] M. Aamir, S. Sharma, and A. Grover, “ChaCha20-in-Memory for Side-Channel Resistance in IoT Edge-Node Devices,” in IEEE Open Journal of Circuits and Systems, vol. 2, pp. 833-842, 2021, doi: 10.1109/OJCAS.2021.3127273.

[2] V. K. Rajanna, S. Taneja and M. Alioto, "SRAM with In-Memory Inference and 90% Bitline Activity Reduction for Always-On Sensing with 109 TOPS/mm2 and 749-1,459 TOPS/W in 28nm," ESSCIRC 2021 - IEEE 47th European Solid State Circuits Conference (ESSCIRC), 2021, pp. 127-130, doi: 10.1109/ESSCIRC53450.2021.9567830.

[3] J. Wang et al., "A 28-nm Compute SRAM With Bit-Serial Logic/Arithmetic Operations for Programmable In-Memory Vector Computing," in IEEE Journal of Solid-State Circuits, vol. 55, no. 1, pp. 76-86, Jan. 2020, doi: 10.1109/JSSC.2019.2939682.

[4] S. Jain, L. Lin and M. Alioto, "Broad-Purpose In-Memory Computing for Signal Monitoring and Machine Learning Workloads," in IEEE Solid-State Circuits Letters, vol. 3, pp. 394-397, 2020, doi: 10.1109/LSSC.2020.3024838.

[5] X. Si et al., "A Twin-8T SRAM Computation-in-Memory Unit-Macro for Multibit CNN-Based AI Edge Processors," in IEEE Journal of Solid-State Circuits, vol. 55, no. 1, pp. 189-202, Jan. 2020, doi: 10.1109/JSSC.2019.2952773.

[6] S. Xu, X. Chen, Y. Wang, Y. Han, X. Qian and X. Li, "PIMSim: A Flexible and Detailed Processing-in-Memory Simulator," in IEEE Computer Architecture Letters, vol. 18, no. 1, pp. 6-9, 1 Jan.-June 2019, doi: 10.1109/LCA.2018.2885752.

[7] Y. N. Wu, J. S. Emer and V. Sze, "Accelergy: An Architecture-Level Energy Estimation Methodology for Accelerator Designs," 2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), 2019, pp. 1-8, doi: 10.1109/ICCAD45719.2019.8942149.

[8] D. Gao, D. Reis, X. S. Hu and C. Zhuo, "Eva-CiM: A System-Level Performance and Energy Evaluation Framework for Computing-in-Memory Architectures," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 5011-5024, Dec. 2020, doi: 10.1109/TCAD.2020.2966484.

[9] M. Poremba, S. Mittal, D. Li, J. S. Vetter and Y. Xie, "DESTINY: A tool for modeling emerging 3D NVM and eDRAM caches," 2015 Design, Automation and Test in Europe Conference and Exhibition (DATE), 2015, pp. 1543-1546, doi: 10.7873/DATE.2015.0733.

[10] Dong, Q., Sinangil, M.E., Erbagci, B., Sun, D., Khwa, W., Liao, H., Wang, Y., & Chang, J. (2020). 15.3 A 351TOPS/W and 372.4GOPS Compute-in-Memory SRAM Macro in 7nm FinFET CMOS for Machine-Learning Applications. 2020 IEEE International Solid- State Circuits Conference - (ISSCC), 242-244.

[11] A. Agrawal, A. Jaiswal, C. Lee and K. Roy, "X-SRAM: Enabling In-Memory Boolean Computations in CMOS Static Random Access Memories," in IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 65, no. 12, pp. 4219-4232, Dec. 2018, doi: 10.1109/TCSI.2018.2848999.

[12] J.W. Su et al., "16.3 A 28nm 384kb 6T-SRAM Computation-in-Memory Macro with 8b Precision for AI Edge Chips," 2021 IEEE International Solid- State Circuits Conference (ISSCC), 2021, pp. 250-252, doi: 10.1109/ISSCC42613.2021.9365984.