A Major Project Phase-II Report

On

"8-BIT ARITHMETIC LOGIC UNIT USING MODIFIED GATE DIFFUSION INPUT TECHNIQUE"

SUBMITTED IN PARTIAL FULFILLMENT OF

THE REQUIREMENTS FOR THE AWARD OF DEGREE OF

BACHELOR OF ENGINEERING

IN

ELECTRONICS & COMMUNICATION ENGINEERING

BY

DIVYESH PANCHASARA	**(1608-20-735-065)**
B. RAHUL	**(1608-20-735-318)**
P.YUVA BHASKAR	**(1608-20-735-321)**

Under the guidance of

Mr. P.RAVIKUMAR REDDY

M.Tech (Ph.D.)

Assistant Professor

Department of Electronics and Communication Engineering

MATRUSRI ENGINEERING COLLEGE

(An Autonomous Institution)

(Sponsored by Matrusri Education Society, Estd1980)

(Approved by AICTE, Affiliated to Osmania University)

16-1-486, Saidabad, Hyderabad, Telangana-500 059

www.matrusri.edu.in

2023-2024

Matrusri Engineering College

(An Autonomous Institution)

(Sponsored by: MATRUSRI EDUCATION SOCIETY, Estd: 1980)

(Approved by AICTE, Affiliated to Osmania University)

email: principal@matrusri.edu.in web site: www.matrusri.edu.in Ph: 040-24072764

Department of Electronics and Communication Engineering

Date: 07/06/2024

Certificate

This is to certify that the Major Project Phase-2 report entitled **"8-BIT ARITHMETIC LOGIC UNIT USING MODIFIED GATE DIFFUSION INPUT TECHNIQUE"** being submitted by **Mr. DIVYESH PANCHASARA (1608-20-735-065), Mr. B. RAHUL (1608-20-735-318), P. YUVA BHASKAR (1608-20-735-321)** in partial fulfilment for the award of the Degree of Bachelor of Engineering in Electronics and Communication Engineering of the Osmania University, Hyderabad, during 2023-24, is a record of bonafide work carried out under our guidance and supervision.

The results presented in this project have not been submitted to any other University or Institute for the award of any Degree or Diploma.

Major Project Phase-II Coordinator
Dr. N. SHRIBALA
Dr. K. KOTESHWARA RAO

Name of the Guide
Mr. P. RAVI KUMAR REDDY
Assistant Professor

Dr. N. SRINIVASA RAO
HOD-ECE

DECLARATION

This is to declare that the work submitted in the present project work report titled **"8-BIT ARITHMETIC LOGIC UNIT USING MODIFIED GATE DIFFUSION INPUT TECHNIQUE"** is a record of bonafide work done by us in the Department of Electronics & Communication Engineering, Matrusri Engineering College, Saidabad, Hyderabad.

No part of the report is copied from books, journals, internet and wherever the subject content is taken, the same has been duly referred to in the text. The report generated is based on the project work carried out entirely by us and not copied from any other source.

Mr. DIVYESH PANCHASARA (1608-20-735-065)
Mr. B. RAHUL (1608-20-735-318)
Mr. P. YUVA BHASKAR (1608-20-735-321)

Place: Hyderabad,
Date: 07/06/2024.

ACKNOWLEDGEMENT

We would like to take this opportunity to place it on the record, that this project would never have taken shape but for the cooperation extended to us by certain individuals. Though it is not possible to name all of them, it would be a pardonable on our part if we don't mention some of the very important people. Sincerely we acknowledge our deep sense of gratitude to the project guide, **Mr. P. RAVI KUMAR REDDY**, Assistant Professor for his constant encouragement, help and valuable suggestions. We wish to thank him for his constant motivation and help throughout the project.

We would like to express our deep gratitude to the Head of Department **Dr. N SRINIVASA RAO** and Project Coordinators **Dr. N. SHRIBALA and Dr. K. KOTESWARA** for their timely cooperation while carrying the project. It is their friendliness that made us feel free and learn more from them.

Last but not the least we would like to thank all those people associated directly or indirectly with the project.

Mr. DIVYESH PANCHASARA (1608-20-735-065)
Mr. B. RAHUL (1608-20-735-318)
Mr. P. YUVA BHASKAR (1608-20-735-321)

ABSTRACT

An 8-bit Arithmetic Logic Unit (ALU) designed using the Modified Gate Diffusion Input (GDI) technique involves creating logical operations and arithmetic functions using a low-power and area-efficient method. MGDI is known for its ability to reduce the number of transistors required for complex logic circuits. In this 8-bit ALU, the MGDI technique optimizes the design by employing a combination of gates to perform arithmetic (addition, subtraction) and logic (AND, OR, NOT, XOR, XNOR) operations. Each bit of the ALU is processed to construct arithmetic functions like addition and subtraction, the MGDI-based ALU uses a combination of logic gates along with multiplexers and carries logic to manage overflow conditions. The MGDI technique offers advantages in reducing transistor count, minimizing power consumption, and optimizing layout area by combining multiple functions in a single gate. Additionally, MGDI-based ALUs tend to exhibit improved performance in terms of speed due to their simplified structure and reduced parasitic capacitance. Overall, the MGDI-based 8-bit ALU balances functionality, performance, and efficiency making it a suitable choice for various low-power applications requiring arithmetic and logical computations within a limited hardware footprint.

Keywords: ALU, GDI, VLSI, Transistor count, Power consumption, Area optimization, Arithmetic operations.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

INTRODUCTION

1.1 Introduction:

In digital logic design, an Arithmetic Logic Unit (ALU) is a fundamental building block that performs various arithmetic and logical operations on binary data. An 8-bit ALU operates on 8-bit binary numbers, handling numbers ranging from 0 to 255 (or -128 to 127 for signed integers).

The MGDI technique, known as Modified Gate Diffusion Input, is a powerful design methodology that leverages the diffusion region of a single transistor to implement multiple logic functions. This leads to several advantages:

- **Reduced transistor count:** Compared to conventional CMOS-based designs, MGDI circuits often require fewer transistors, resulting in a smaller area footprint and lower power consumption.
- **Simplified design process:** MGDI offers a more intuitive and concise approach to circuit design, as it directly maps logic functions to transistor configurations.
- **Improved performance:** Depending on the specific implementation, MGDI circuits can exhibit lower power consumption and fewer delays.

MGDI is a design methodology that focuses on reducing transistor count and power consumption by leveraging the diffusion capacitance of transistors. Unlike traditional CMOS designs, MGDI achieves this by cascading gates and sharing common nodes, thereby simplifying the overall transistor configuration. The advantages of MGDI are significant. Firstly, it leads to more compact circuit designs, as fewer transistors are required to implement complex logic functions. This reduction in transistor count directly translates to a smaller chip area, which is crucial for optimizing space in modern integrated circuits. Additionally, MGDI circuits consume less power due to the reduced number of active transistors, making them ideal for applications where power efficiency is paramount. Moreover, MGDI circuits often exhibit faster switching speeds compared to traditional designs, further enhancing their appeal in high-performance computing environments. However, designing circuits using the MGDI technique also presents challenges. It requires careful planning and

consideration of gate arrangements and signal routing to ensure proper functionality and signal integrity.

1.2 Problem statement

The current landscape of digital circuit design faces challenges in achieving an optimal balance between power efficiency, speed, and area utilization in the construction of Arithmetic Logic Units (ALU). Hence, There is a need to explore alternative design techniques that offer improvements in these key aspects.

1.3 Objective

The objective is to design an 8-bit Arithmetic Logic Unit (ALU) using the Modified Gate Diffusion Input (MGDI) technique to perform arithmetic and logical operations efficiently while optimizing the transistor-level implementation.

1.4 Specifications

The proposed ALU would perform the respective operations as mentioned below. Each of these operations would be implemented with the help of the select lines of the multiplexer.

- **Arithmetic Operations:** Addition, Subtraction.
- **Logical Operations:** AND, OR, XOR, NOT, Bitwise AND, Bitwise OR, Bitwise XOR, Bitwise NOT.

1.5 Methodology

1.5.1 Requirements Analysis and Architecture Design:

- **Define the functionality:** List all the operations your ALU needs to perform.
- **Identify performance requirements:** Specify power consumption limitations.
- **Choose an architecture:** Select a suitable architecture for your ALU, considering factors like modularity, scalability, and ease of implementation with MGDI.
- **Break down the architecture:** Divide the ALU into smaller functional units like adders, multiplexers, and control logic.
- **Design and Testing of Sub Circuits Logics:** After designing the basic logic gates integrate the basic gates to form various logic such as adder, subtractor, and multiplexer.

- **SubSystem Design and Testing:** Once the successful testing of the sub-circuits is done integrate these circuits to design other hierarchical blocks.

1.5.2 Simulation and Verification:

- **Choose appropriate tools:** Select simulation tools and libraries for MGDI circuits.
- **Develop test vectors:** Create comprehensive test vectors to cover all functionalities and corner cases.
- **Simulate the design:** Perform simulations under various operating conditions and analyze results for correctness and performance.
- **Verify functionality:** Use formal verification techniques or additional test methods to ensure the design meets all requirements.

1.6 Motivation.

- **Gain a deeper understanding of MGDI:** By delving into MGDI's principles and applying them to a practical design, you can solidify your knowledge and gain valuable insights into its capabilities and limitations.
- **Explore alternative design techniques:** Comparing MGDI with traditional CMOS design approaches can broaden your perspective and highlight the potential advantages and trade-offs involved.
- **Contribute to research and development:** Document the findings of the project that could contribute to the ongoing research and development of MGDI technology.

1.7 Layout of Thesis

The thesis investigates the design and implementation of an 8-bit Arithmetic Logic Unit (ALU) using the Modified Gate Diffusion Input (MGDI) technique. The Introduction chapter emphasizes the significance of ALUs in digital electronics and the need for enhanced design techniques. The Literature Review compares traditional CMOS technology, Gate Diffusion Input (GDI), and MGDI, highlighting MGDI's superior performance. In the Design Methodology, the requirements analysis, architecture design, and a detailed step-by-step process are discussed. The Implementation chapter provides extensive schematics and circuit diagrams. Simulation and Results obtained using LTspice in 65nm technology demonstrate significant improvements in power efficiency and area reduction with MGDI compared to CMOS and GDI.

Chapter 2

LITERATURE SURVEY

2.1 Introduction

The Arithmetic Logic Unit (ALU) is a fundamental building block of digital electronics, particularly within the central processing unit (CPU) of a computer. Its design and efficiency are crucial as they directly influence the performance and power consumption of microprocessors. Recent advancements have focused on optimizing ALU design to reduce power consumption, transistor count, and area, while improving speed and efficiency. Various innovative techniques such as Modified Gate Diffusion Input (m-GDI) have been proposed and implemented to achieve these goals. This literature survey explores multiple studies that delve into different ALU design methodologies, emphasizing the use of m-GDI and other advanced techniques to enhance the performance of ALUs in digital systems.

2.2 Literature Review:

[1]. Aiyyappu Surendrababu N.Ashokkumar. "Design of ALU using 2T XOR Gate and Decoder", International Journal on Cybernetics & Informatics (IJCI) Vol. 10, No.3, June 2023.

This study proposes an Arithmetic and Logic Unit (ALU) design with a modified Gate Diffusion Input (m-GDI), as well as an ALU design with a 2T XOR GATE and a DECODER. The usage of modified Gate Diffusion Input in these techniques minimizes the delay path, power consumption, area, and transistor count. The power consumption is exceptionally low, yielding a smaller footprint and a shorter delay path. These strategies rely heavily on the 2T XOR Gate in Digital VLSI Design, which is based on Full Adder and Full Subtractor. The circuit for a 1-to-8 decoder was designed for several logic families before being implemented in an 8-bit Arithmetic and Logic Unit.

[2]. Anil Nageshwar Rangapure, Dr.Kiran. "8-bit Arithmetic Logic Unit Design using Modified Gate Diffusion Input (m-GDI) Technique" .International Journal of Advances in Engineering and Management (IJAEM), Volume 3, Issue 9 Sep 2021, ISSN: 2395-5252.

This study presents the design of an 8-bit ALU using the modified Gate Diffusion Input (MGDI) approach. The arithmetic logic unit may be designed with a significantly lower transistor count and very low power consumption by utilizing the GDI approach leading to decreased chip space and power consumption, the two most crucial factors in digital VLSI design. In this design, the FA (full adder) uses three transistor XORs. Additionally, a 1-to-8 deMUX circuit was incorporated into the design. After examining and comparing a small number of research papers with logic families, an 8-bit ALU that can execute 8 different types of operations is ultimately developed.

[3]. Samuel Winchenbach, Mohammed Driss. "8-bit Arithmetic Logic Unit." University of Maine, Orono, 2019, Available online: https://ece.umaine.edu/wp-content/uploads/sites/203/2012/05/ALU.pdf

The design, layout, and testing of an 8bit Arithmetic Logic Unit (ALU) based on the 74LS181 4bit ALU circuit from Texas Instruments is discussed. Two 4bit ALUs were cascaded together to create one 8bit ALU on chip. Details of the design process are included from each step in the project, along with diagrams, photos, and renders showing the various components and testing setups. The project was undertaken as a means to learn the VLSI design process and resulted in a fully functional ALU with the same functionality as a commercially available product. Minimal problems were encountered with the final design and were addressed

[4]. P. Sai Krishna, P. Brundavani. "Design and Implementation of Low Power 16-Bit ALU using MGDI Technique", ISSN 2322-0929, Vol.05, Issue.10, October-2019, Pages:0954-0959.

Numerous research efforts are prompted by the rapid growth of portable digital applications and the need for ever-increasing speed, small implementation, and low power dissipation. Over the last 20 years, a number of logic design methodologies have been developed in response to the desire to enhance the performance of logic circuits that were previously built using conventional CMOS technology. Modified Gate Diffusion Input, or MGDI, is a low-power digital combinational design approach that uses fewer transistors while allowing for lower power consumption and propagation delay when compared to other commonly used logic design techniques. This work presents and compares a 16-bit arithmetic logic

unit with CMOS logic utilizing the MGDI technique. The addition, subtraction, increment, and deduction operations, AND, OR, XOR, and XNOR, are the arithmetic and logical functions that are realized in the ALU.

[5]. Harshmaniyadav Uday Panwar. "Design of 8-Bit ALU Design using GDI Techniques with Less Power and Delay". International Journal of Recent Technology and Engineering (IJRTE), Volume-8 Issue-4, November 2017. ISSN: 2277-3878

The microchip's Arithmetic Logic Unit (ALU) is a significant component. Modern computer processors and mathematical operations are all accomplished by using the ALU. This study presents an 8-bit ALU that uses a Gate Dispersion Input (GDI) centered MUX and the lowest power 11-transistor Full Adder (11-T FA). Tanner EDA software version-15 with 32 nanometer BSIM4 innovation was used to model all structures. In order to retain both power and delay, an 8-bit ALU operating in the subthreshold region of 0.7 VDD was used in this research. Because of the voltage level enhancement, the GDI suggested the model's 8-bit ALU consumed less than 82% less power than the CMOS 8-bit ALU.

[6]. Dr. K. Srinivasulu, M. Shiva Kumar, Chetempally Sridhar Goud. "Design of a Low Power Area Efficient ALU Using Modified GDI Multiplexer." October 3 2017.

The main concern in low voltage and low power applications is the optimization of area, delay, and power dissipation. Digital combinational design with minimal power consumption is GDI-Gate Diffusion Technology. Comparatively speaking, this technique uses fewer transistors than traditional CMOS technology. Optimization of area and power is attained when the number of transistors is decreased. Poor logic swing is one of GDI's drawbacks, however it may be fixed by altering this method. The optimized area and low-power 8-bit ALU employing a Modified Gate Diffusion Input Multiplexer are the major topics of this work. With a complete logic swing, this method permits a decrease in area, latency, and low power dissipation. Full adder is a basic cell in ALU that is designed using XOR-MUX to have an operation with high speed and low power. The entire design is done using CADENCE Tool in GPDK 45nm technology. Power and delay comparison between conventional CMOS, GDI, and Modified GDI is also presented.

[7]. N.Srinu, M.Kedhar, O.Ajay, "Design of Low-Power ALU Using GDI Technique." 2018 IJCRT, Volume 6, Issue 2 April 2016, ISSN: 2320-2882.

The purpose of this paper is to design low-power and area-efficient ALU using the GDI technique. The main sub-modules of ALU are adder, logical unit, Subtractor, multiplexer, and divider. This work evaluates and compares the performance & optimized area of ALU with conventional CMOS style & GDI technique the simulations are performed by using a Micro wind tool. At first, by using the Micro Wind tool the circuits are designed & simulated with the CMOS technique and then with the GDI technique. By comparing the two designs GDI and CMOS style then GDI has the advantage of less power and less area.

[8]. Anitesh Sharma, Ravi Tiwari. "Low Power 8-Bit ALU Design Using Full Adder and Multiplexer". International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE 2016.

The arithmetic logic unit (ALU) is an important part of the microprocessor. In the digital processor, logical and arithmetic operations are executed using ALU. In this paper, we describe 8-bit ALU using low power 11-transistor full adder (FA) and Gate diffusion input (GDI) based multiplexer. By using FA and multiplexer, we have reduced the power and delay of 8-bit ALU as compared to the existing design. All designs were simulated using DSCH and Microwind 3.5 in 65 nm BSIM4 technology.

[9]. A.S. Prabhu, B. Naveena, K. Parimaladevi, M. Samundeswari, P. Thilagavathy. "Serial Divider Using Modified GDI Technique." International Journal of Innovative Research in Electrical (IJIRE). Vol. 3, Issue 10, October 2016.

Divider is a basic hardware module in advanced and high-speed digital signal processing (DSP) units. It has applications in radar technology, communication, industrial control systems, and linear predictive coding (LPC) algorithms in speech processing. This paper presents a 4-bit Serial Divider using the Modified GDI Technique. The repeated one's complement method of the binary subtraction algorithm is used for serial division. The proposed method aims at Modified Gate Diffusion Input (mod-GDI) which is a low-power technique to design any digital system. This technique has been adopted from Gate Diffusion Input (GDI). The

Modified GDI technique allows for reducing delay, power consumption, and area compared to conventional CMOS Technology. According to the estimations done, the transistor count, Tool analysis time, power consumption of serial divider using CMOS technology, and the Modified GDI technique are tabulated. The designs are simulated using the Tanner EDA tool.

[10]. Pinninti Kishore, K. Babulu and P. V. Sridevi. "Low Power and High-Speed Carry Save Adder Using Modified Gate Diffusion Input Technique." ARPN Journal of Engineering and Applied Sciences. VOL. 11, NO. 21, NOVEMBER 2016, ISSN 1819-6608.

Low-power and high-speed adders are the most essential components of every contemporary signal processing application. Among the many adders, Carry Save Adder (CSA) is the high-speed multi-operand adder used in many applications. This paper presents the design of a low-power and high-speed Carry Save Adder with a reduced no. of transistors using the Modified Gate Diffusion Input (MGDI) technique. This technique has been adopted from the Gate Diffusion Input Technique (GDI) and is used to achieve reduced power consumption, delay, and area of digital circuits while maintaining the low complexity of logic design. This paper aims to design 4 operands 8-bit and 16-bit CSA using conventional CMOS, GDI, and MGDI techniques. After various performance comparisons, it is stated that total power dissipation, propagation delay, and transistor count are much smaller in the MGDI technique than compared to CMOS and GDI techniques.

2.3 Conclusion

The literature reviewed highlights significant strides in ALU design, primarily through the application of Modified Gate Diffusion Input (MGDI) techniques. These studies consistently demonstrate that MGDI can substantially reduce power consumption, transistor count, and area, while also improving the speed of ALUs. Comparisons with conventional CMOS technology reveal the superiority of MGDI in achieving low power and high efficiency.

Chapter 3

PROBLEM SPECIFICATIONS

3.1 Introduction

An 8-bit Arithmetic Logic Unit (ALU) designed using the Modified Gate Diffusion Input (MGDI) technique aims to provide a low-power and area-efficient solution for performing arithmetic and logical operations. The MGDI technique significantly reduces the number of transistors required for complex logic circuits, optimizing the design through the use of gates such as NAND, NOR, XOR, and XNOR. The ALU must support essential arithmetic operations like addition and subtraction, as well as logical operations including AND, OR, and NOT. Each bit of the ALU processes arithmetic functions by employing a combination of XOR, AND, and OR gates, supplemented by multiplexers, and carry logic to handle overflow conditions effectively. Logical operations leverage NOR, NAND, and inverter gates to achieve the desired functionalities. The MGDI technique's ability to combine multiple functions within a single gate offers significant advantages, including reduced transistor count, minimized power consumption, and optimized layout area. This design approach also enhances the ALU's performance by simplifying its structure and reducing parasitic capacitance, thereby improving operational speed.

The ALU must operate within specified power supply ranges, meeting required speed and operational frequency standards. Inputs include binary operands and an operation selector (opcode), while outputs encompass the result. Performance metrics will focus on power consumption and area utilization. The design process involves detailed schematic creation, layout optimization, and comprehensive simulation and testing to ensure the ALU meets all functional and performance criteria.

Ultimately, the MGDI-based 8-bit ALU aims to balance functionality, performance, and efficiency, making it an ideal choice for low-power applications that require arithmetic and logical computations within a constrained hardware footprint.

3.2 Design Constraints

- Technology: Use MGDI to leverage its advantages in low power and area efficiency.
- Power Consumption: Optimize the design to ensure minimal power usage.

- Area: Minimize the chip area by reducing the number of transistors and optimizing the layout.
- Transistor Count: Reduce the number of transistors for each operation to benefit from the MGDI technique.
- Power Supply: Ensure the ALU operates within the specified power supply range for MGDI circuits.
- Input/Output Specifications:
    - Inputs: Binary operands (A, B), operation selector (opcode)
    - Outputs: Result and Carry-out.

3.3 Block Diagram of 8-bit ALU.

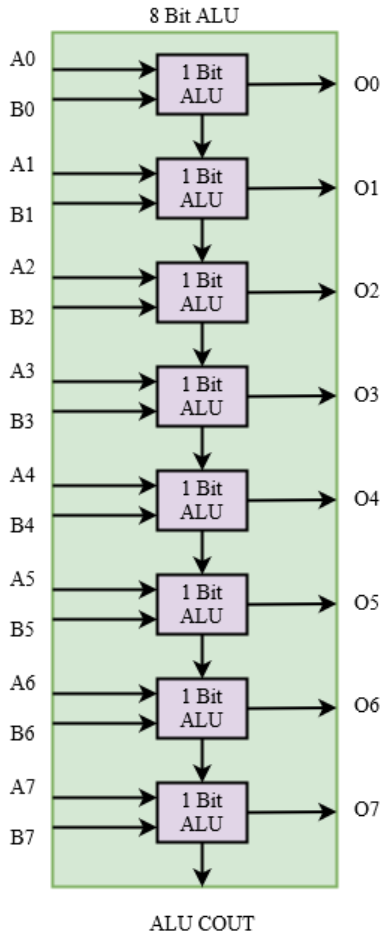

Fig 3.1: Architecture of 8-Bit ALU.

The Fig 3.1 represents a multi-bit ALU constructed by connecting several 1-bit ALUs in a series. This design approach enables the ALU to handle multi-bit operations, where each bit of the input operands is processed by an individual 1-bit ALU. The multi-bit ALU consists of multiple stages, each stage corresponding to a single bit of the operands. For instance, an 8-bit ALU would be composed of eight

1-bit ALUs. Each 1-bit ALU processes a pair of bits from the input operands (A and B) along with a carry-in bit, and select lines, and it generates a sum bit and a carry-out bit.

In this configuration, each 1-bit ALU operates on its respective bit in parallel with the other 1-bit ALUs. However, the presence of the carry bit introduces a sequential dependency. The carry-out from each 1-bit ALU must be propagated to the carry-in of the next higher significant bit's 1-bit ALU. This means that while the bitwise operations are performed in parallel, the overall operation of the multi-bit ALU has a ripple effect due to the carry propagation. This characteristic means that the multi-bit ALU can be considered semi-parallel because it must wait for the carry bits to propagate through all the stages.

The inputs to the multi-bit ALU include the multi-bit operands A and B, and an initial carry-in bit, usually set to zero for addition operations. The outputs consist of the multi-bit result and a final carry-out bit, which indicates if there was an overflow in the case of addition. The sequential propagation of carry bits makes this design more like a ripple-carry adder, which is simple and effective for small bit-widths.

3.3.1 Internal Structure of a 1-bit ALU

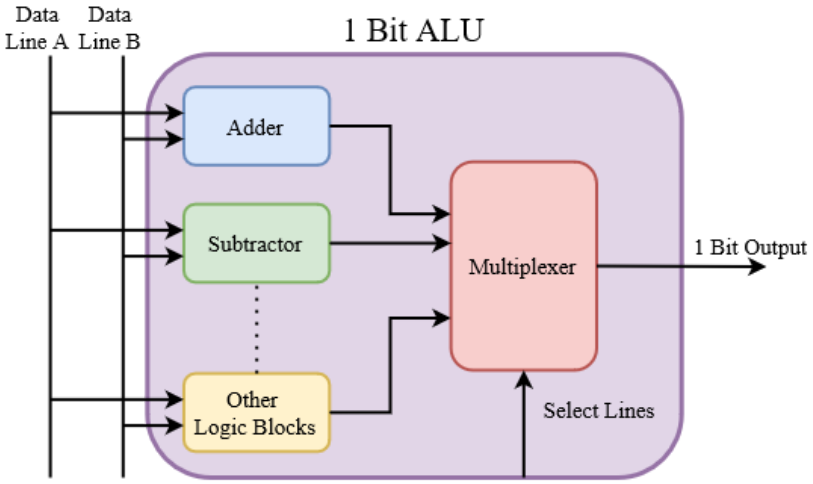

Fig 3.2: Architecture of 1-Bit ALU.

The Fig 3.2 provides a view of the internal structure of a single 1-bit ALU. Each 1-bit ALU is responsible for performing arithmetic and logical operations on individual bits of the operands. The core components of the 1-bit ALU include an adder, a subtractor, various logic blocks, and a multiplexer (MUX).

The adder performs bitwise addition on the input bits from operands A and B, along with a carry-in bit. It outputs a sum bit and a carry-out bit. The carry-out bit is essential for operations on multi-bit ALUs as it propagates to the next significant bit's 1-bit ALU. The subtractor similarly handles bitwise subtraction, producing a difference bit and potentially a borrow bit, depending on the operands' values.

In addition to arithmetic operations, the 1-bit ALU includes other logic blocks to perform fundamental logical operations such as AND, OR, XOR, XNOR, CLEAR and NOT. These operations are crucial for many computational tasks beyond basic arithmetic. The multiplexer (MUX) is a critical component that selects the output of the 1-bit ALU based on control (select) lines. The MUX receives inputs from the adder, subtractor, and other logic blocks, and the select lines determine which operation's result is outputted by the ALU. This design allows the ALU to be versatile, and capable of performing different operations based on the control signals provided.

When considering the multi-bit ALU composed of several 1-bit ALUs, it becomes clear that while each 1-bit ALU processes its respective bit of the operands simultaneously, the design's overall operation is influenced by the sequential carry propagation. This hybrid approach—parallel bitwise operations combined with serial carry propagation—strikes a balance between simplicity and functionality. Each 1-bit ALU’s internal structure, equipped with an adder, subtractor, logic blocks, and a multiplexer, ensures that the ALU can perform a variety of arithmetic and logical operations, making it a fundamental building block in digital computing.

3.3.2 Operations Performed By The 8-Bit ALU

The Arithmetic Logic Unit (ALU) is a fundamental component of digital circuits, responsible for executing a variety of arithmetic and logical operations. The specific operations performed by an ALU include:

- **Addition:** The ALU can perform addition of two 8-bit operands, A and B, with an optional carry-in (CIN). This operation computes the sum of A, B, and CIN, producing an 8-bit result and a carry-out if the sum exceeds the 8-bit limit.
- **Subtraction:** The ALU can subtract one 8-bit operand (B) from another (A), incorporating a borrow-in (BIN). This operation calculates the difference (A -

B - BIN), producing an 8-bit result and a borrow-out if the subtraction requires borrowing.

- **Logical AND:** The ALU performs a bitwise AND operation on two 8-bit operands, A and B.
- **Logical OR:** The ALU executes a bitwise OR operation on two 8-bit operands, A and B.
- **Logical XOR:** The ALU performs a bitwise exclusive OR (XOR) operation on two 8-bit operands, A and B.
- **Logical XNOR:** The ALU executes a bitwise exclusive NOR (XNOR) operation on two 8-bit operands, A and B.
- **Logical NOT:** The ALU can perform a bitwise NOT operation on a single 8-bit operand, A.
- **Clear A:** The ALU includes an operation to clear the value of operand A, setting all bits to zero.

Table 3.1 : Operations Performed By 8 BIT ALU.

OPCODE	OPERATION PERFORMED
000	CLEAR A
001	ADD (A + B + CIN)
010	AND (A & B)
011	OR (A | B)
100	SUB (A - B - BIN)
101	XOR (A ^ B)
110	XNOR ~(A ^ B)
111	NOT A

Chapter 4

DESIGN OF LOGIC BLOCKS

4.1 Design of MGDI CELL

Modified Gate Diffusion Input (MGDI) is a technique employed in the design of digital circuits to improve performance metrics such as power consumption, speed, and area. It is an evolution of the standard Gate Diffusion Input (GDI) method, which aims to simplify circuit design and enhance its efficiency. It consists of a NMOS transistor and a PMOS transistor.

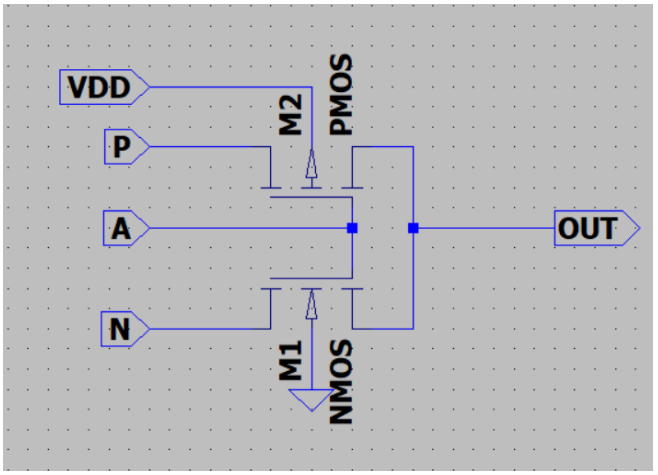

Fig 4.1: MGDI CELL.

TruthTable of MGDI:

In an MGDI circuit, there are three inputs A, N, and P and one output, Y. The gate of the PMOS transistor is connected to input A, its source to input P, and its bulk to VDD, while the gate of the NMOS transistor is also connected to input A, its source to input N, and its bulk to GND. The drains of both transistors are connected to form output Y.

Table 4.1: Truth Table of MGDI CELL.

A	N	P	OUT
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0

A	N	P	OUT
1	0	1	0
1	1	0	1
1	1	1	1

Physical Dimensions:

- NMOS: Length: 65nm, Width: 65nm.
- PMOS: Length: 65nm, Width: 162.5nm.

Working of MGDI Cell

The working principle of an MGDI cell is based on the states of its three inputs: A, N, and P. The cell comprises an NMOS transistor and a PMOS transistor, with their gates connected to the input AWhen input A is low (0), the PMOS transistor turns on because it is activated by a low gate voltage, while the NMOS transistor turns off because it is deactivated by a low gate voltage. As a result, the output Y will follow the input P: if P is 1, Y will be 1; if P is 0, Y will be 0. Conversely, when input A is high (1), the PMOS transistor turns off and the NMOS transistor turns on. Consequently, the output Y will follow the input N: if N is 1, Y will be 1; if N is 0, Y will be 0. Thus, when the input at both gates is low, the output is determined by the P port, and when the input at both gates is high, the output is determined by the N port.

Key Features of MGDI

- Low Power Consumption
- High-Speed Operation
- Reduced Area
- Simplified Design

4.2 Design of Basic Gates

Using the foundational Modified Gate Diffusion Input (MGDI) cell, we can construct a full range of essential logic gates that are crucial for digital circuit operations. The MGDI technique employs a minimalistic design, requiring only one PMOS and one NMOS transistor per gate. This approach offers significant benefits, including lower power consumption, reduced silicon area, and faster operation, making it highly suitable for low-power, high-performance applications. In the

sections that follow, we will detail the design of the seven basic logic gates AND, OR, NAND, NOR, XOR, XNOR, and NOT using the MGDI methodology. Each of the gates may require one or more MGDI cells to meet the specifications. Ones the basic gates are designs any logics can be obtained by the combination of these gates. Hence, it is must to design and test the gates properly to meet the requirements. If any requirements were not to be reached the gates was remodeled and designed to perform the task efficiently as, these basic gates is the foundation for the Digital Design or any other circuit design.

4.2.1 Design of AND Gate

An AND gate is a fundamental building block in digital electronics. It is a type of logic gate that implements the logical conjunction operation. In an AND gate, the output is high (1) only when all its inputs are high (1). If any input is low (0), the output is low (0). AND gates are also critical in arithmetic circuits, such as adders and multipliers. In binary adders, AND gates are used to compute the carry-out bit, which is essential for multi-bit binary addition. This role is especially important in the design of full adders and ripple-carry adders used in processors and other computational hardware. Overall, the versatility and simplicity of AND gates make them indispensable in various digital electronics applications, from basic logic functions to sophisticated computational systems.

Input : A, B

Output : Y = A.B

Truth Table:

Table 4.2: Truth Table of AND Gate.

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

MGDI AND Gate:

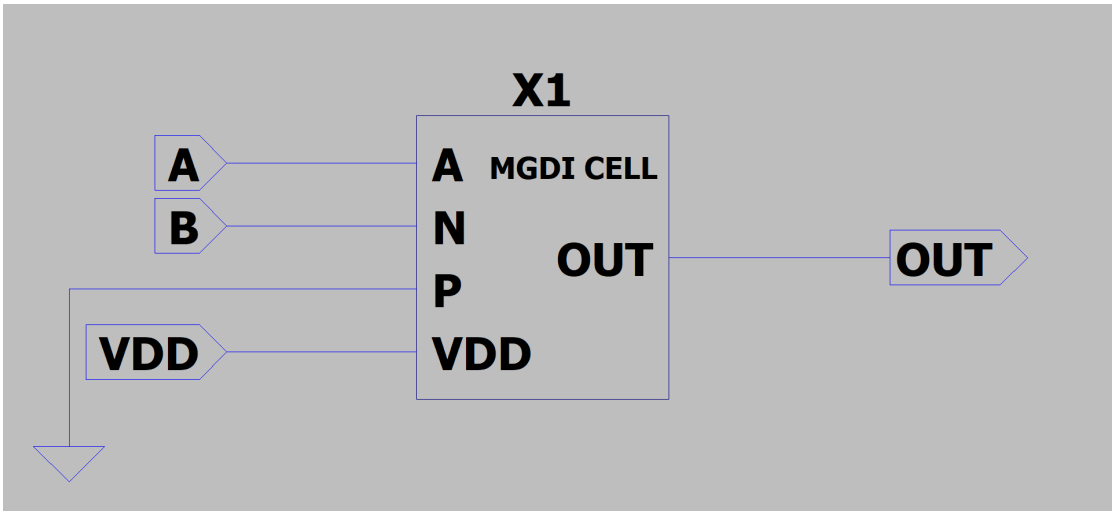

Fig 4.2 : MGDI AND Gate.

To design an AND gate using the Modified Gate Diffusion Input (MGDI) technique, we configure the MGDI cell with specific connections. In this configuration, the P port of the MGDI cell is set to 0 (GND), the N port is connected to input B, and the gate of both transistors is connected to input A. The output is taken from the common drain of the PMOS and NMOS transistors. When input A is high, the NMOS transistor is active, and the output depends on input B. If B is also high, the NMOS transistor conducts, pulling the output low. Conversely, if either A or B is low, the output remains high, fulfilling the AND logic function.

4.2.2 Design of OR Gate

An OR gate is a fundamental component in digital electronics, performing the logical disjunction operation. It outputs a high signal (1) if at least one of its inputs is high (1). If all inputs are low (0), the output is low (0). The OR gate is crucial in building more complex digital systems and performing decision-making processes in circuits. There are some simple OR gates that take only one input but there are also OR gates that take more than one input they are known as Multi-Input OR Gate.

Technically, you can have an OR logic gate with any number of inputs. The output will be true if any of the N inputs are true. Here, we are discussing the two common types of OR gates. Also known as Basic OR logic gate, this type of OR gate takes in two input values to produce a single output value. Here there are two input values, hence the possible combination of inputs will be 4.

Input : A, B

Output : Y = A + B

Truth Table

Table 4.3: Truth Table of OR Gate.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

MGDI OR Gate:

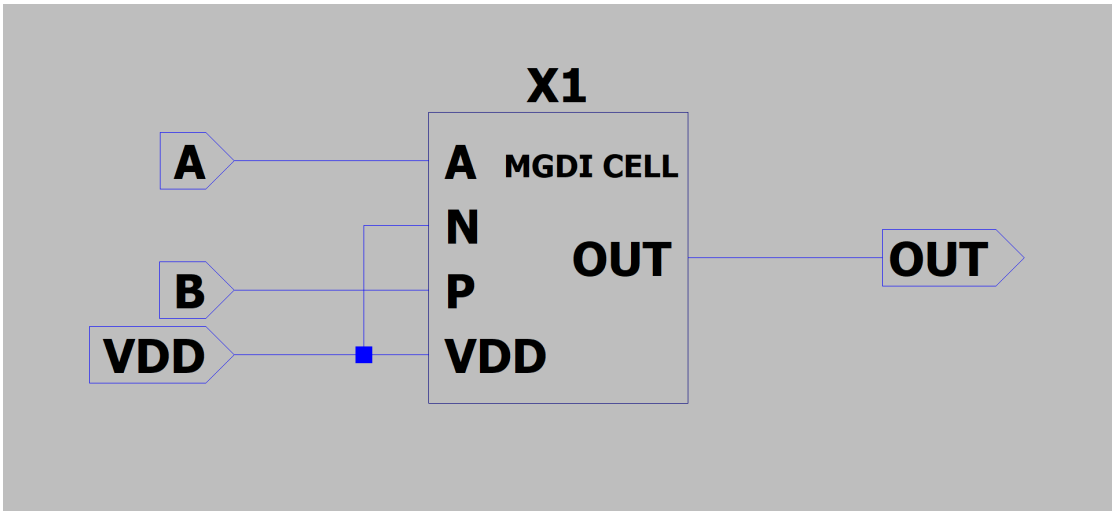

Fig 4.3 : MGDI OR Gate.

To design an OR gate using the Modified Gate Diffusion Input (MGDI) technique, we set up the MGDI cell with the following connections: the P port is connected to input B, the N port is set to 1 (VDD), and the gate of both the PMOS and NMOS transistors is connected to input A. The output is taken from the common drain of the PMOS and NMOS transistors. When input A is low, the PMOS transistor is active, and the output is pulled high if B is high. When input A is high, the NMOS transistor is active, and since N is always 1 (VDD), the output is also pulled high regardless of B's value.

4.2.3 Design of NAND Gate

A NAND gate is a fundamental building block in digital electronics, performing the logical NAND (Not AND) operation. It is essentially an AND gate followed by a NOT gate. The NAND gate outputs a low signal (0) only when all its inputs are high (1). If any input is low (0), the output is high (1). This gate is significant because it is functionally complete, meaning any logical function can be implemented using only NAND gates.

Input : A, B

Output : Y = ~(A.B)

Truth Table

Table 4.4: Truth Table of NAND Gate.

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

MGDI NAND Gate:

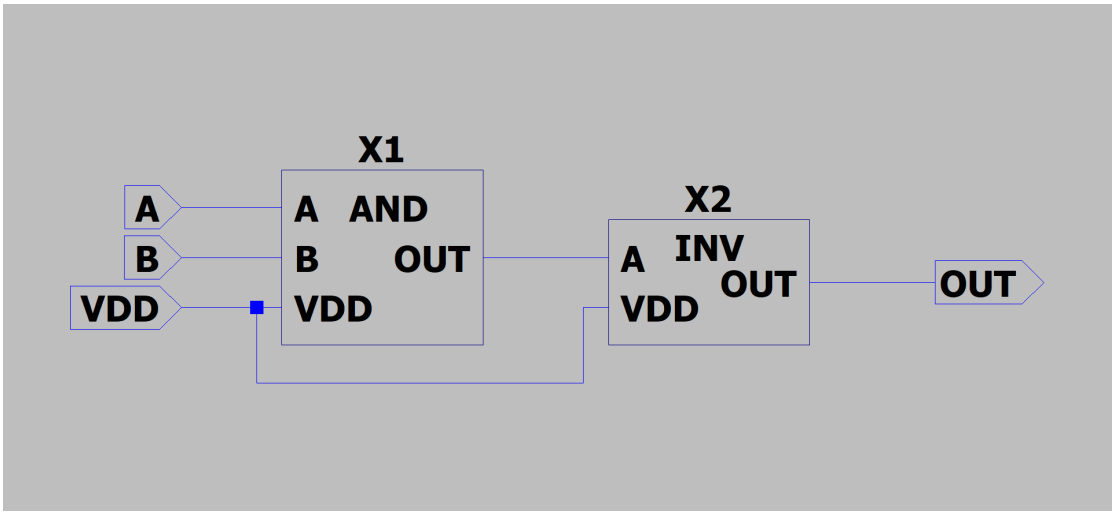

Fig 4.4 : MGDI NAND Gate.

To design a NAND gate using the Modified Gate Diffusion Input (MGDI) technique, we start with the configuration of the MGDI cell for an AND gate and then connect an additional NOT gate to the output. For the MGDI AND gate, the P port is set to 0 (GND), the N port is connected to input B, and the gate of both the PMOS and NMOS transistors is connected to input A. The output of this AND gate configuration provides the AND logic.

To achieve the NAND function, we need to invert the output of the AND gate. This is done by connecting a NOT gate to the output of the MGDI AND gate. The NOT gate inverts the output, so when the AND gate output is high (indicating both A and B are high), the NOT gate will output low, and when the AND gate output is low (indicating either A or B is low), the NOT gate will output high.

4.2.4 Design of NOR Gate

A NOR gate is a fundamental digital logic gate that performs the logical NOR (Not OR) operation. It is essentially an OR gate followed by a NOT gate. The NOR gate outputs a high signal (1) only when all its inputs are low (0). If any input is high (1), the output is low (0). Like the NAND gate, the NOR gate is functionally complete, meaning any logical function can be implemented using only NOR gates.

Input : A, B

Output : Y = ~(A.B)

Truth Table

Table 4.5: Truth Table of NOR Gate.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

MGDI NOR Gate:

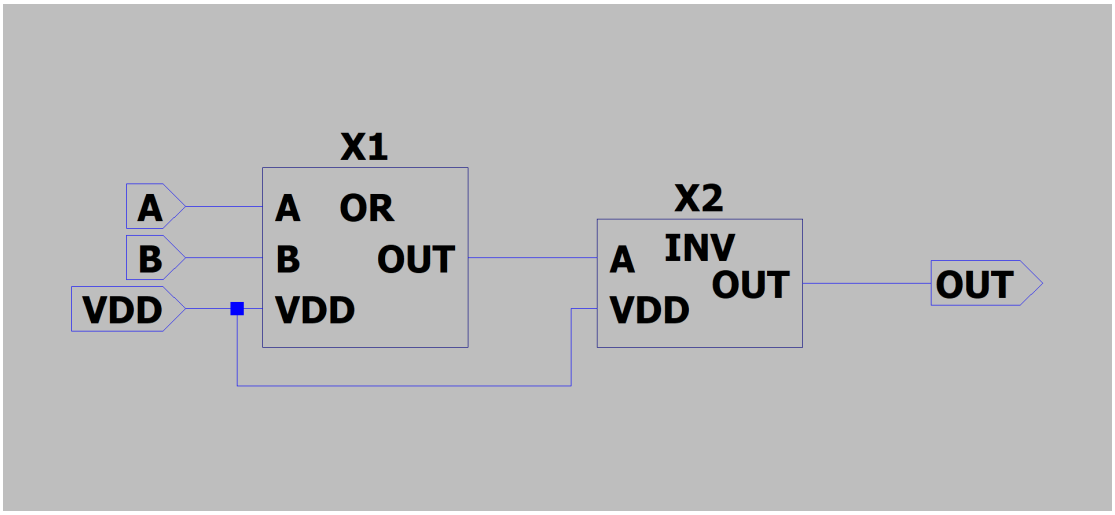

Fig 4.5 : MGDI NOR Gate.

To design a NOR gate using the Modified Gate Diffusion Input (MGDI) technique, we begin by configuring the MGDI cell as an OR gate and then incorporate an additional NOT gate to invert its output. In the OR gate configuration, the P port is connected to input B, the N port is set to 1 (VDD), and the gate of both the PMOS and NMOS transistors is connected to input A. The output is taken from the common drain of the PMOS and NMOS transistors. This setup ensures that the output is high if either input A or B is high. To achieve the NOR functionality, we need to invert this output, which is accomplished by connecting a NOT gate to the OR gate's output. The NOT gate inverts the high output of the OR gate to low, and the low output to high. Therefore, the final output is high only when both inputs A and B are low, effectively implementing the NOR logic gate with the efficiency and simplicity of the MGDI technique.

4.2.5 Design of XOR Gate

An XOR (exclusive OR) gate is a fundamental digital logic gate that performs the exclusive OR operation. It outputs a high signal (1) if an odd number of its inputs are high (1). For a 2-input XOR gate, the output is high (1) if only one of the inputs is high (1). If both inputs are low (0) or both are high (1), the output is low (0). XOR gates are crucial in arithmetic operations, error detection, and digital logic design.

Input : A, B

Output : Y = A.B’ + A’.B

Truth Table

Table 4.6: Truth Table of XOR Gate.

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

MGDI XOR Gate:

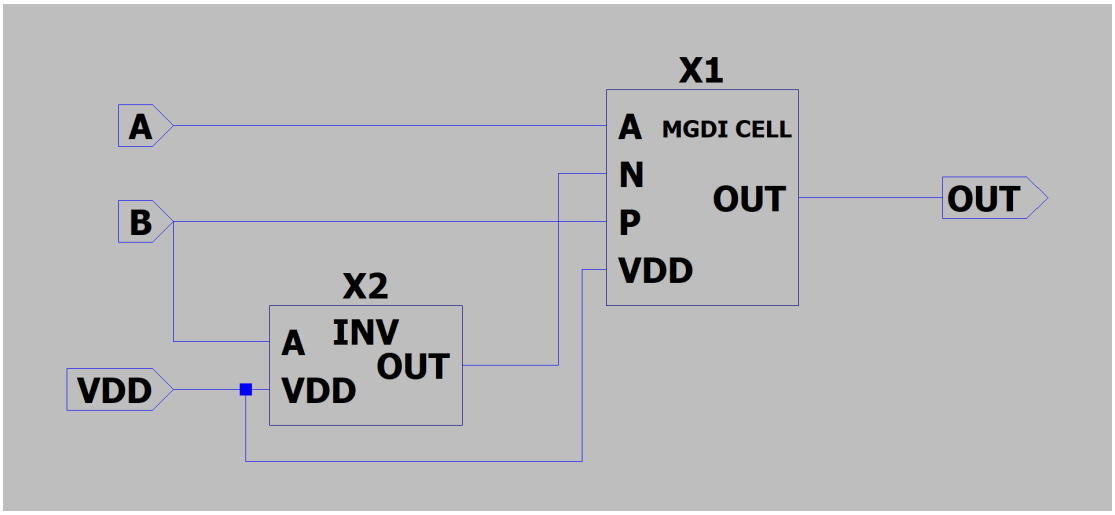

Fig 4.6 : MGDI XOR Gate.

To implement an XOR gate using the Modified Gate Diffusion Input (MGDI) technique, we employ a single MGDI cell, with the input signal B directly connected to the P port and the complement of B (B') connected to the N port via an inverter. This setup ensures that when input A is high, the output of the MGDI cell depends on the values of B and its complement, B'. Specifically, if A is high and B is low (B' is high), or if A is low and B is high (B' is low), the output will be high, indicating an XOR logic operation. Conversely, if both A and B have the same value, the output will be low.

4.2.6 Design of XNOR Gate

An XNOR (exclusive NOR) gate is a fundamental digital logic gate that performs the logical XNOR operation, which is the complement of the XOR

operation. It outputs a high signal (1) if an even number of its inputs are high (1). For a 2-input XNOR gate, the output is high (1) if both inputs are the same (both 0 or both 1). If one input is high (1) and the other is low (0), the output is low (0). XNOR gates are used in equality checking and other digital logic applications where equivalence is required.

Input : A, B

Output : Y = A.B + A'.B'

Truth Table

Table 4.7: Truth Table of XNOR Gate.

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

MGDI XNOR Gate:

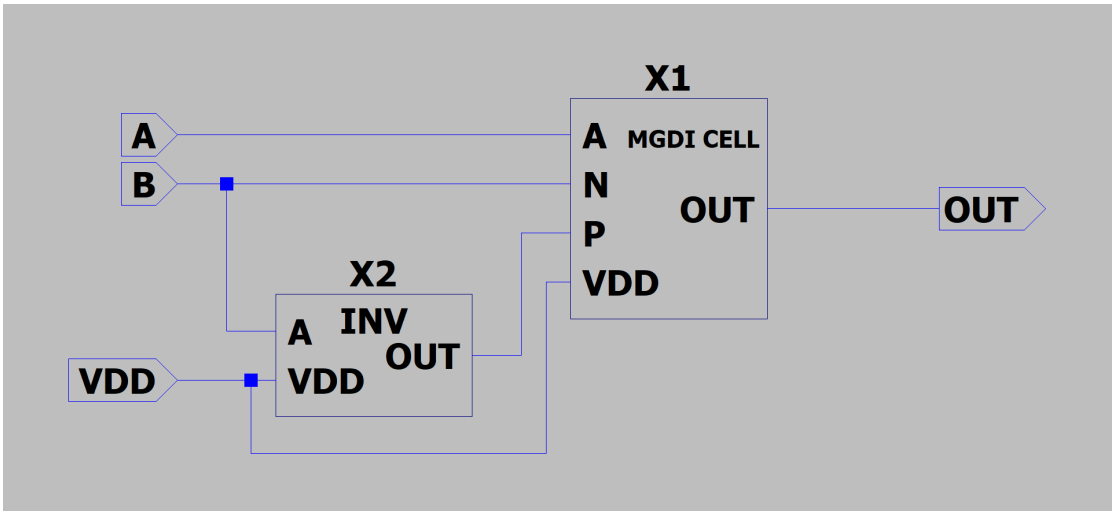

Fig 4.7 : MGDI XNOR Gate.

To implement an XNOR gate using the Modified Gate Diffusion Input (MGDI) technique, we configure a single MGDI cell with the input signal B connected to the N port and the complement of B (B') connected to the P port via an inverter. This setup ensures that when input A is high, the output of the MGDI cell

depends on the values of B and its complement, B'. Specifically, if A is high and B is high (B' is low), or if A is low and B is low (B' is high), the output will be high, indicating an XNOR logic operation. Conversely, if both A and B have different values, the output will be low.

4.2.7 Design of NOT Gate

A NOT gate, also known as an inverter, is a fundamental digital logic gate that performs the logical negation operation. It inverts the input signal; if the input is high (1), the output is low (0), and if the input is low (0), the output is high (1). The NOT gate is essential in digital circuits for implementing logical complement operations and is widely used in combination with other logic gates to create complex logic functions.

Input : A

Output : Y = A'

Truth Table

Table 4.8: Truth Table of NOT Gate.

A	Y
0	1
1	0

MGDI NOT Gate:

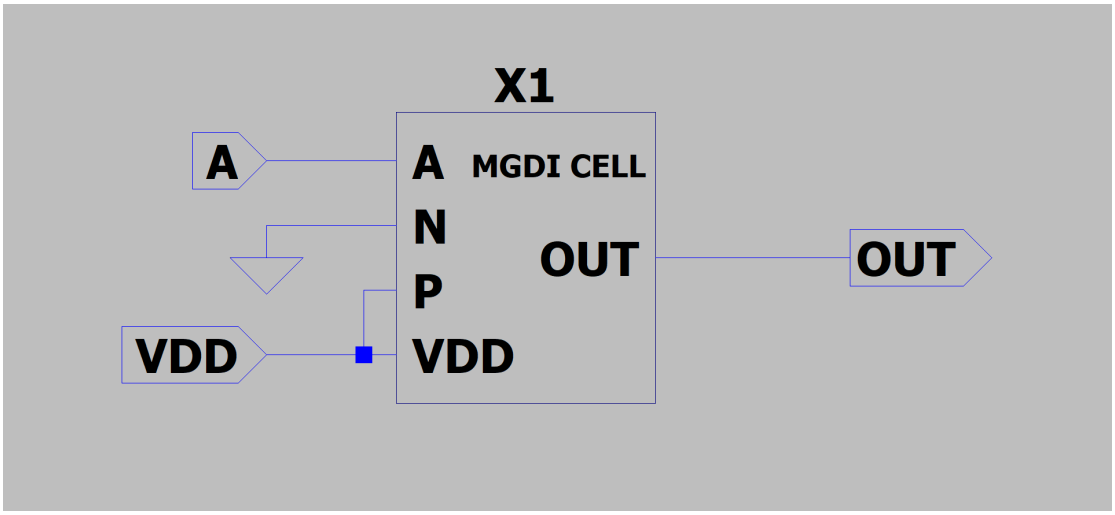

Fig 4.8 : MGDI NOT Gate.

To implement a NOT gate using Modified Gate Diffusion Input (MGDI) technology, you need to connect the P port of the MGDI cell to the supply voltage (VDD) and the N port to the ground (GND). In this configuration, the input signal (A) is applied to the gate of both the P-type and N-type transistors within the MGDI cell, making it common to both gates

4.3 Design of Combinational Logic

Designing combinational logic circuits using the Modified Gate Diffusion Input (MGDI) technique offers a streamlined approach to achieving efficient and reliable digital systems. Combinational logic circuits, which produce outputs based solely on the current inputs without considering previous input states, are fundamental building blocks in digital design. The MGDI methodology simplifies the design process by utilizing a minimalistic approach, often requiring only one PMOS and one NMOS transistor per gate. This simplicity leads to reduced power consumption, smaller silicon area, and enhanced speed performance. In this context, we explore how to leverage the principles of MGDI to design various combinational logic circuits, including basic gates, arithmetic circuits, multiplexers, and decoders.

4.3.1 Design of Half Adder.

A half adder is a fundamental digital circuit used in arithmetic operations within computers and other digital systems. It is designed to add two single-bit binary numbers and produce a sum and a carry output. The half adder forms the basis for more complex adders, such as full adders, which can handle multiple-bit additions.
The functionality of the half adder can be described by the following equations:

- **Sum (S)**: The sum output is the result of the XOR (exclusive OR) operation on the inputs A and B. **S=A^B**.
- **Carry (C)**: The carry output is the result of the AND operation on the inputs A and B. **C=A.B**

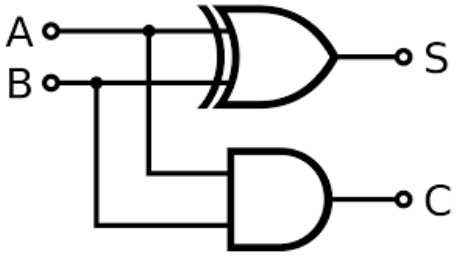

Fig 4.9 : Half Adder Logic Diagram.

Input: A, B

Output: Sum = A ^ B, Carry = A.B

Truth Table:

Table 4.9: Truth Table of Half Adder.

A	B	Difference	Borrow
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

MGDI Half Adder:

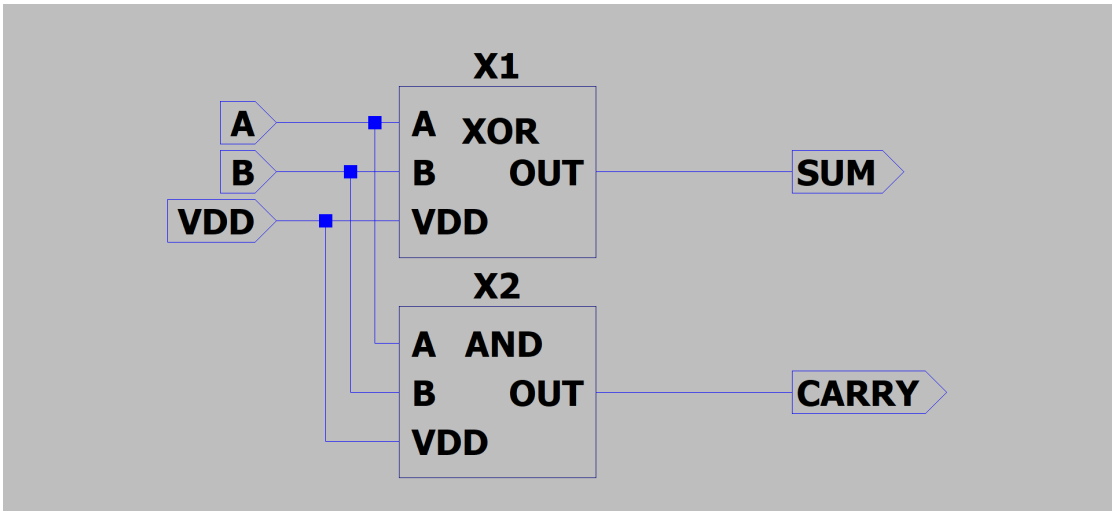

Fig 4.10 : Half Adder Schematic LTSPICE.

To design a half adder using the Modified Gate Diffusion Input (MGDI) technique and basic logic gates such as AND and XOR gates, we follow a systematic approach. First, we design the AND gate and the XOR gate using the MGDI methodology. For the AND gate, we configure the MGDI cell with one input connected to the P port and the other to the N port, with the output taken from the common drain. For the XOR gate, we utilize an MGDI cell with one input connected to both the P and N ports and the other input connected to the inverted signal via an inverter. Once these gates are designed, we combine them to form the half adder. The

output of the AND gate represents the carry-out, indicating whether there is a carry from the addition of the two input bits. The output of the XOR gate represents the sum, providing the result of the binary addition without considering any carry.

4.3.2 Design of Full Adder.

A full adder is a crucial digital circuit used in binary arithmetic operations, specifically for adding binary numbers. Unlike a half adder, which can only handle the addition of two single-bit binary numbers, a full adder can add three single-bit binary numbers: two significant bits and an additional carry-in bit from a previous stage of addition. This capability makes the full adder an essential building block for constructing multi-bit binary adders, such as those used in arithmetic logic units (ALUs) and digital processors.

Full adders are crucial components in binary addition within arithmetic logic units (ALUs) of microprocessors and digital signal processors (DSPs). They are also integral in designing multi-bit adders such as ripple-carry adders, where multiple full adders are cascaded to add binary numbers of greater bit lengths. Additionally, full adders are used in digital circuits for performing subtraction (using two's complement arithmetic) and in various arithmetic operations required in digital computing and signal processing. They play a vital role in error detection and correction schemes, such as those used in communication systems, by facilitating the generation and verification of parity bits.

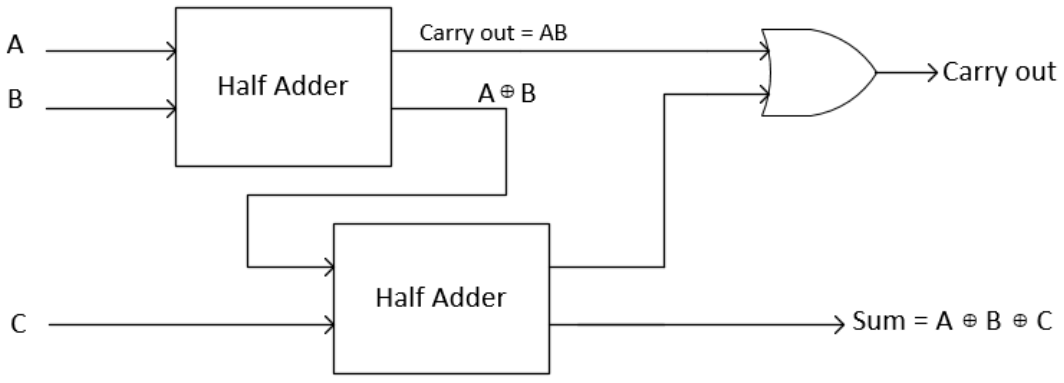

Fig 4.11 : Full Adder Logic Diagram.

Input: A, B , Cin

Output: D = A ^ B ^ Cin, Bout = A.B + (A ^ B)

Truth Table:

Table 4.10: Truth Table of FullAdder.

A	B	Bin	Sum	Carry
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

MGDI Full Adder:

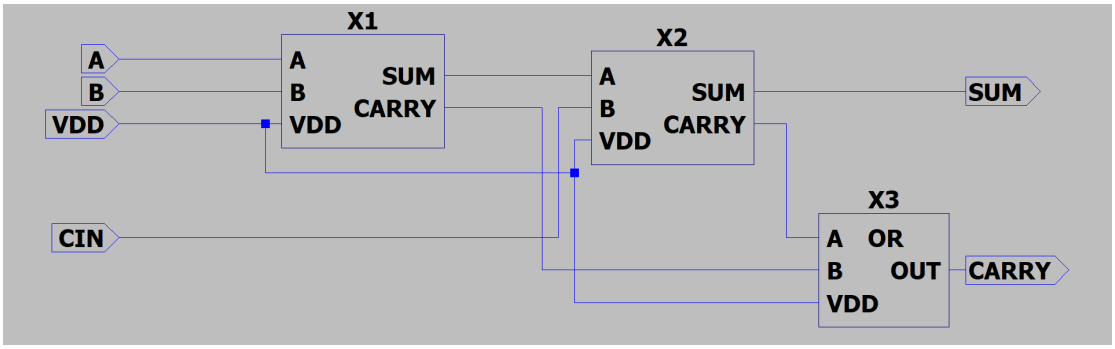

Fig 4.12 : Full Adder Schematic LTSPICE.

To design a full adder using the Modified Gate Diffusion Input (MGDI) technique, we first construct two half adders, which serve as the foundational components. Each half adder computes the sum (S) and carry-out (Cout) for two input bits. The output of the first half adder, which is the sum (S), is then fed as one of the inputs to the second half adder, alongside the input carry (Cin). Subsequently, the second half adder calculates the final sum output. Meanwhile, the carry-out (Cout) generated by the first half adder and the carry-out from the second half adder are fed into an OR gate to compute the overall carry output.

4.3.3 Design of Half Subtractor.

A half subtractor is a basic digital circuit used in binary subtraction operations. It subtracts two single-bit binary numbers and produces a difference and a borrow output. The half subtractor is fundamental in digital systems for performing arithmetic operations, particularly in the context of subtraction.

The functionality of the half subtractor can be described by the following equations:

- **Difference (D)**: The difference output is the result of the XOR (exclusive OR) operation on the inputs A and B. **D=A^ B**
- **Borrow (Bo)**: The borrow output is the result of the AND operation on the negation of A and B. **Bo= A’ · B**

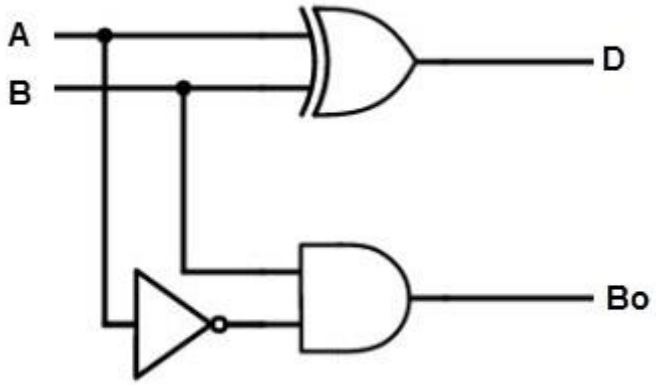

Fig 4.13: Half Subtractor Logic Diagram.

Input: A, B

Output: D = A ^ B, Bout = A' B

Truth Table:

Table 4.11: Truth Table of Half Subtractor.

A	B	Difference	Borrow
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

MGDI Half Subtractor:

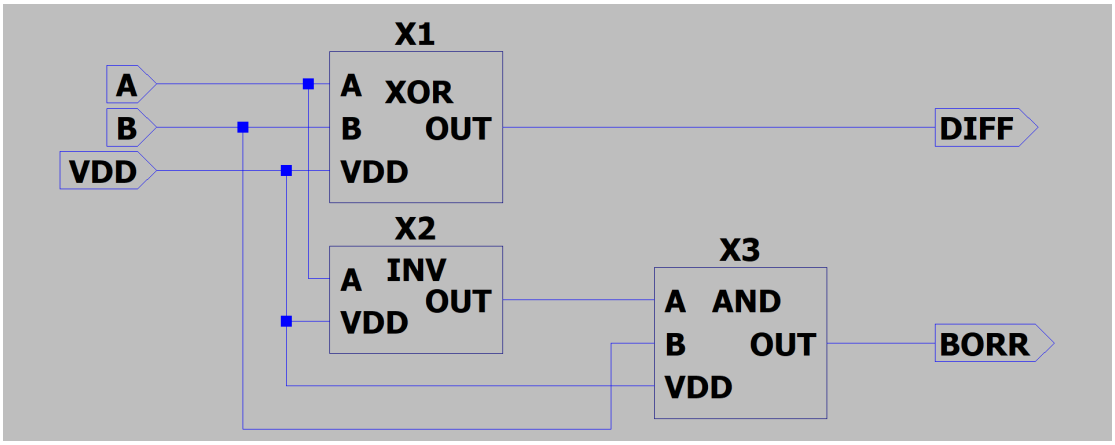

Fig 4.14: Half Subtractor Schematic LTSPICE.

To design a half subtractor using the Modified Gate Diffusion Input (MGDI) technique, we employ basic logic gates while focusing on the core functionality of computing the difference (D) and borrow (Bout) outputs. The XOR gate computes the difference, taking inputs A and B directly. The resulting output provides the difference (D). Additionally, the OR gate, which determines the borrow (Bout), takes inverted input A and input B. This setup ensures Bout is generated when A is 0 and B is 1. By combining the outputs from the XOR and OR gates, we obtain the final outputs of the half subtractor: the difference (D) and the borrow (Bout).

4.3.4 Design of Full Subtractor.

A full subtractor is a fundamental digital circuit used in binary subtraction operations, specifically designed to subtract three single-bit binary numbers: two significant bits and a borrow-in bit from a previous stage. Unlike a half subtractor, which can only handle the subtraction of two single-bit binary numbers, a full subtractor accounts for an additional borrow-in, making it essential for constructing multi-bit binary subtractors in digital systems.

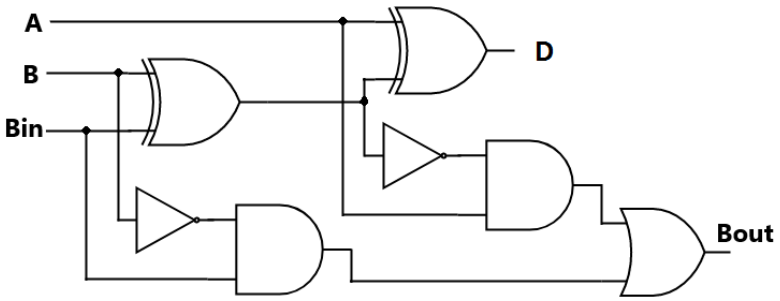

Fig 4.15: Full Subtractor Logic Diagram.

Input: A, B , Bin

Output: D = A ^ B ^ Bin, Bout = A' Bin + A' B + B Bin

Truth Table:

Table 4.12: Truth Table of FullSubtractor.

A	B	Bin	Difference	Borrow
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

MGDI Full Subtractor:

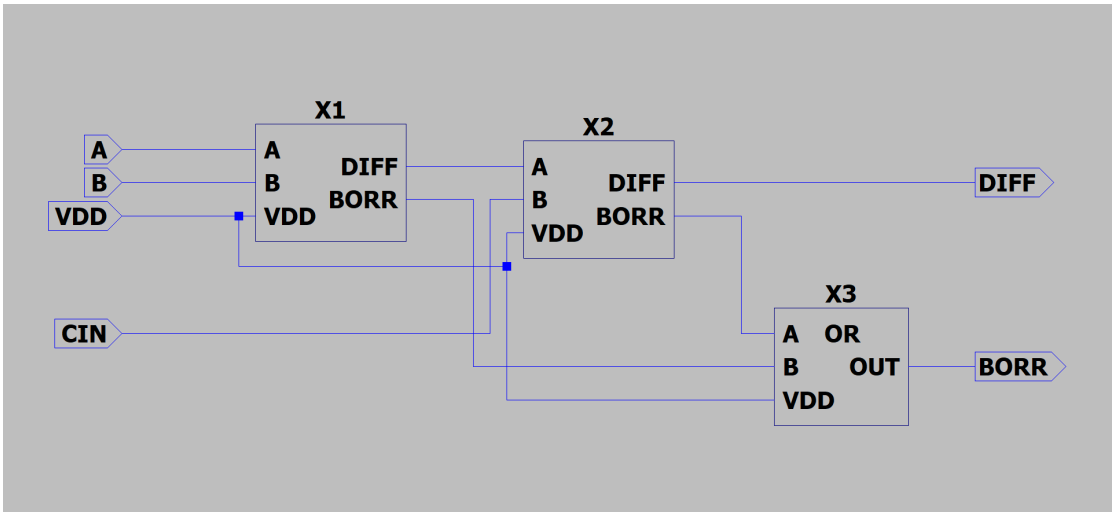

Fig 4.16: Full Subtractor Schematic LTSPICE.

To design a full subtractor using the Modified Gate Diffusion Input (MGDI) technique, we proceed in a step-by-step manner. Initially, we construct the first half

subtractor, where the inputs are A and B, and the difference output is passed to the next stage. Then, for the second half subtractor, the inputs are B and the difference from the previous half subtractor. The output of the second half subtractor represents the final difference. The borrow output is generated by an OR gate, which takes the borrow outputs from both half subtractors as inputs. This configuration ensures that the borrow is generated when there is a borrow from either or both half subtractors.

4.3.5 Design of 2:1 Mux.

A 2:1 multiplexer (MUX) is a fundamental digital circuit used in various applications to select one of two input signals and forward the selected input to a single output line. This selection is controlled by a single binary control signal. Multiplexers are essential components in digital systems, enabling efficient data routing and signal selection within circuits.

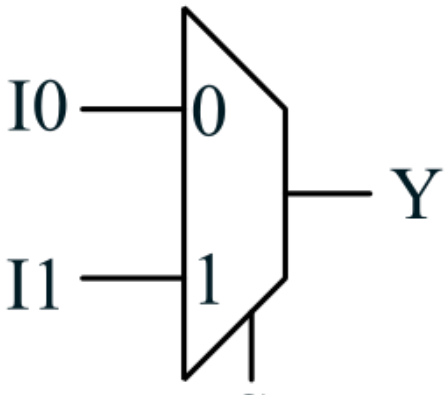

Fig 4.17 : 2 : 1 MUX Logic Diagram.

Input: S0

Output: Y = S0'.I0 + S0.I1

Truth Table:

Table 4.13: Truth Table of 2 : 1 MUX.

S0	Y
0	I0
1	I1

MGDI 2 : 1 MUX:

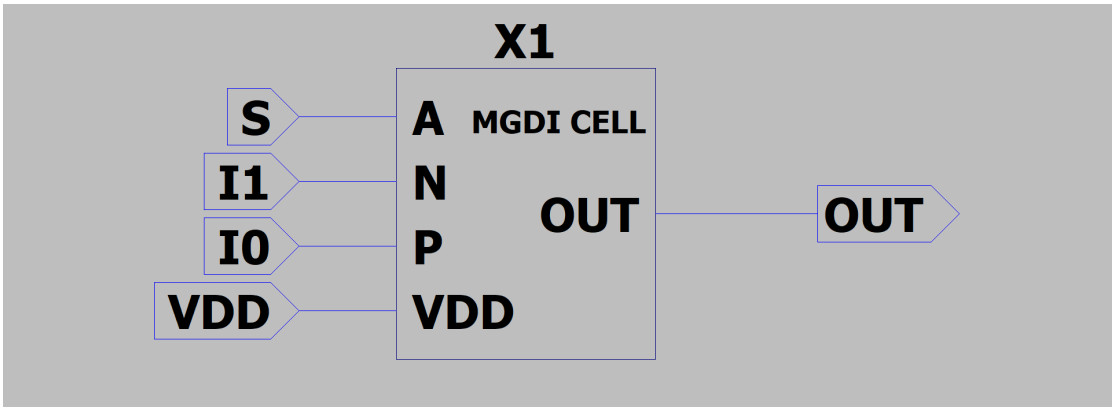

Fig 4.18 : 2 : 1 MUX Schematic LTSPICE.

To design a 2:1 multiplexer (mux) using the Modified Gate Diffusion Input (MGDI) technique, we establish a configuration where one of two inputs is selected based on a control signal. In this setup, one input is connected to the P port of the MGDI cell, while the other input is linked to the N port. The common input acts as the select signal, determining which input is propagated to the output. When the select input is low, the NMOS transistor within the MGDI cell is off, enabling the PMOS transistor to conduct and allowing the first input to pass through. Conversely, when the select input is high, the PMOS transistor is off, permitting the NMOS transistor to conduct, thereby transmitting the second input to the output. The output is obtained from the common drain of the PMOS and NMOS transistors within the MGDI cell.

4.3.6 Design of 4:1 Mux.

A 4:1 multiplexer (mux) is a digital circuit that selects one of four input data signals based on two control signals and forwards it to the output. By using 2:1 multiplexers as building blocks, a 4:1 mux can be effectively constructed. This approach demonstrates the scalability and modularity of digital circuits. Multiplexers are crucial in digital electronics for efficient data routing and management. They simplify the design of digital circuits by enabling multiple signals to share a single data path, reducing the complexity and cost of wiring and components.

By efficiently managing multiple input signals and directing them to a single output, multiplexers are integral components in the design and operation of complex digital systems.

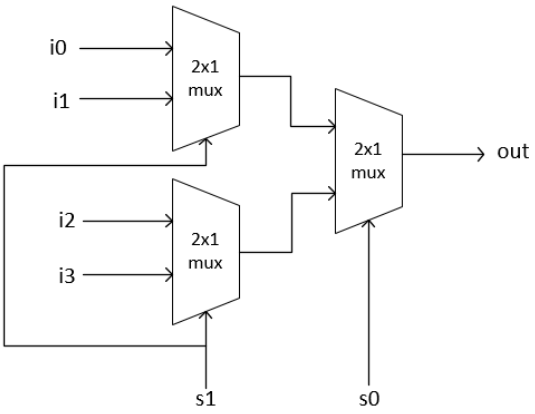

Fig 4.19 : 4 : 1 MUX Logic Diagram.

Input: S1, S0

Output: Y = S1'.S0'.I0 + S1'.S0.I1 + S1.S0'.I2 + S1.S0.I3

Truth Table:

Table 4.14: Truth Table of 4 : 1 MUX.

S1	S0	Y
0	0	I0
0	1	I1
1	0	I2
1	1	I3

MGDI 4 : 1 MUX:

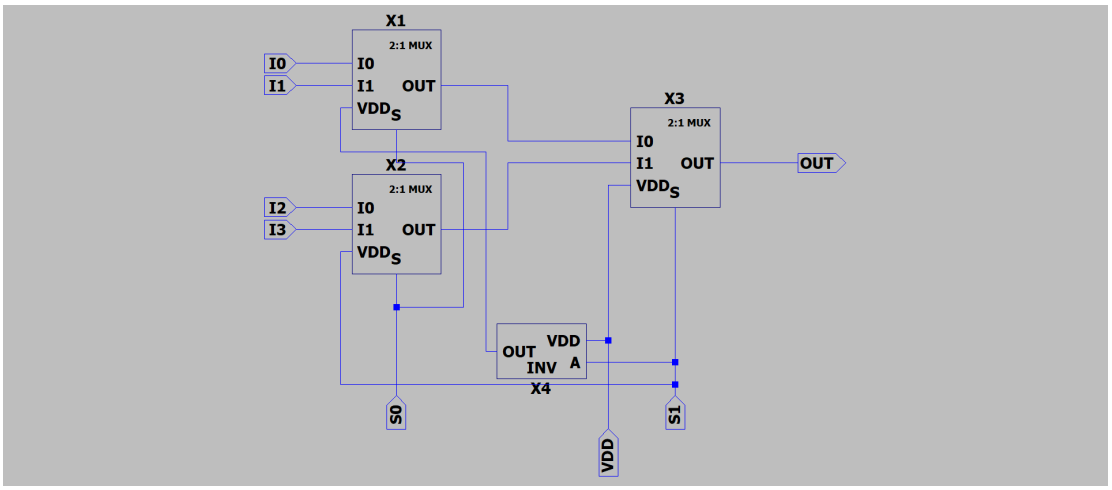

Fig 4.20 : 4 : 1 MUX Schematic LTSPICE.

To construct a 4:1 multiplexer (mux) using the Modified Gate Diffusion Input (MGDI) technique, we utilize three 2:1 muxes organized hierarchically. In the first stage, two 2:1 muxes are employed to select between pairs of inputs. Each mux takes two inputs and uses a select signal to determine which one is passed to the output. Subsequently, in the second stage, a third 2:1 mux is employed to select between the outputs of the first two muxes. The select input of this third mux controls the selection process. By cascading these muxes in this manner, we efficiently route the desired input from the four available inputs to the output based on the select signals. The output of the final mux represents the selected input.

4.3.7 Design of 8:1 Mux.

An 8:1 multiplexer (mux) is a digital circuit that selects one of eight input data signals based on three control signals and forwards it to a single output. This type of multiplexer is essential in digital systems for efficiently managing multiple data lines and is widely used in applications such as data routing, communication systems, and microprocessors. These will works in as to select ans perform a respective operation for the ALU.

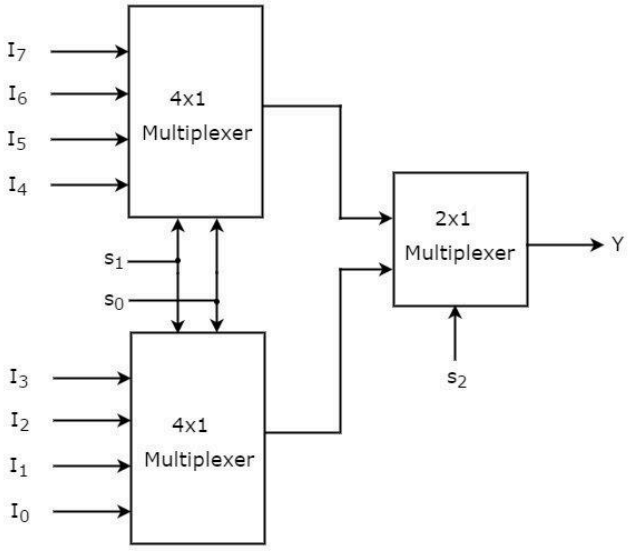

Fig 4.21 : 8 : 1 MUX Logic Diagram.

Input: S2, S1, S0

Output: **Y** = S2'.S1'.S0'.I0 + S2'.S1'.S0.I1 + S2'.S1.S0'.I2 + S2'.S1.S0.I3 + S2.S1'.S0'.I4 + S2.S1'.S0.I5 + S2.S1.S0'.I6 + S2.S1.S0.I7

Truth Table:

Table 4.15: Truth Table of 8 : 1 MUX.

S2	S1	S0	Y
0	0	0	I0
0	0	1	I1
0	1	0	I2
0	1	1	I3
1	0	0	I4
1	0	1	I5
1	1	0	I6
1	1	1	I7

MGDI 8 : 1 MUX:

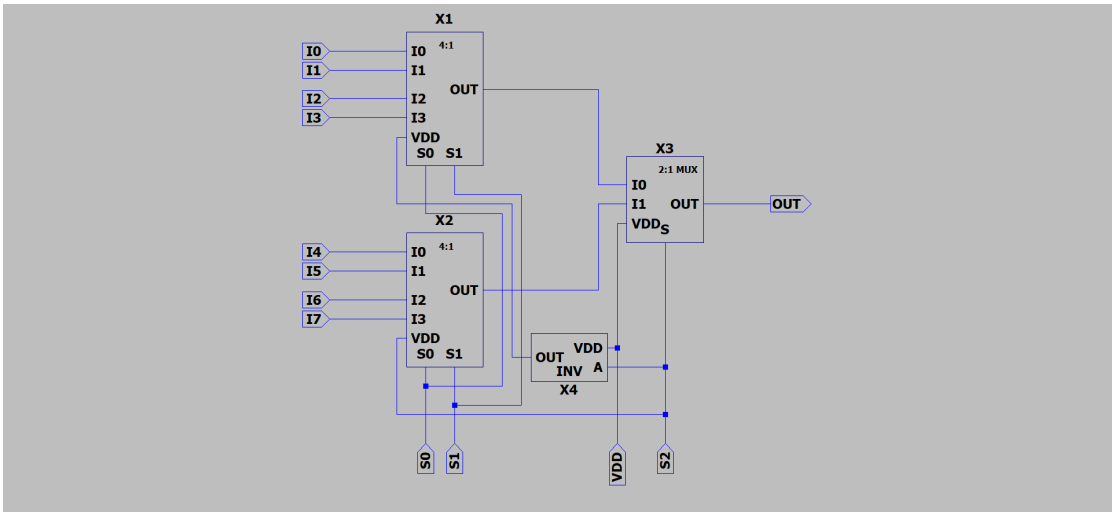

Fig 4.22 : 8 : 1 MUX Schematic LTSPICE.

To construct an 8:1 multiplexer (mux) using the Modified Gate Diffusion Input (MGDI) technique, we employ a hierarchical arrangement of two 4:1 muxes and one 2:1 mux. In the first stage, each 4:1 mux selects between four inputs, with two select inputs, A and B, controlling the selection process. The inputs are grouped

accordingly, with the first 4:1 mux handling inputs 0 to 3 and the second 4:1 mux handling inputs 4 to 7. Subsequently, in the second stage, a 2:1 mux is utilized to select between the outputs of the two 4:1 muxes. The third select input, C, controls this selection process. By cascading these muxes, we efficiently route the desired input from the eight available inputs to the output based on the select signals. The output of the final mux represents the selected input.

4.3.8 Design of 2:4 Decoder.

A decoder is a combinational digital circuit that converts binary information from N input lines to a maximum of 2^N unique output lines. Essentially, a decoder performs the inverse operation of a multiplexer, transforming encoded inputs into a specific, distinct output. It is widely used in various digital systems for tasks such as data demultiplexing, memory address decoding, and implementing control functions. A 2:4 decoder is a digital circuit that decodes a 2-bit binary input into one of four outputs, with each output corresponding to one of the possible combinations of the input bits. It is a fundamental component in digital systems, used for tasks such as memory addressing, data multiplexing, and demultiplexing. Decoders are essential components in digital electronics, enabling the efficient translation of binary inputs into distinct outputs for a variety of applications, enhancing the functionality and performance of digital systems. Decoders are also instrumental in enabling instruction decoding in CPUs, allowing the control unit to interpret and execute various commands efficiently. Additionally, they are used in display systems, such as seven-segment displays, where a binary input needs to be converted to a specific pattern to represent numerical digits.

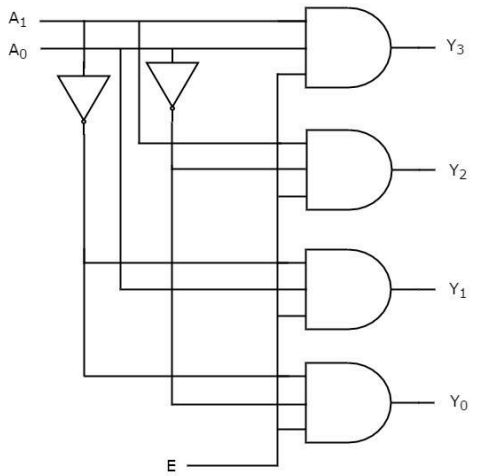

Fig 4.23 : 2 : 4 Decoder Logic Diagram.

Input: I1, I0

Output: Y0, Y1, Y2, Y3

Truth Table:

Table 4.16: Truth Table of 2 : 4 Decoder.

I1	I0	Y0	Y1	Y2	Y3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

MGDI 2 : 4 Decoder:

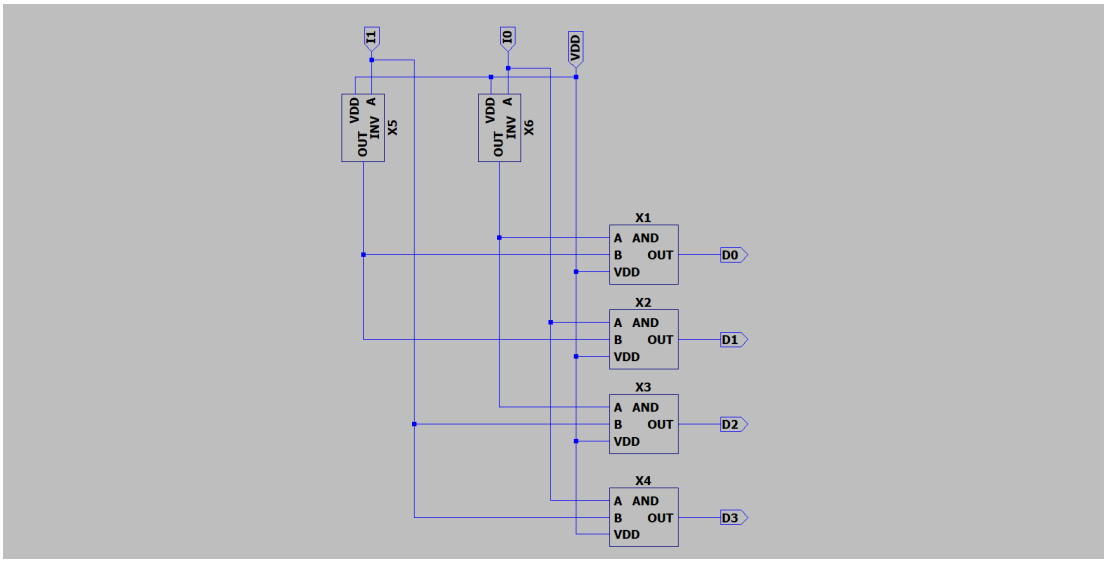

Fig 4.24 : 2 : 4 Decoder Schematic LTSPICE.

In constructing the 2-to-4 decoder, we rely on gate-level logic, employing 2 NOT gates and 4 AND gates. The NOT gates facilitate the inversion of input signals, essential for generating complementary inputs for the AND gates. Each AND gate combines these inputs to produce the corresponding output lines based on the specific input combination. This modular approach enables efficient decoding of the input code into one of four possible outputs, ensuring minimal hardware complexity while maintaining versatility in digital systems.

4.3.9 Design of 3:8 Decoder.

A 3:8 decoder is a digital circuit that decodes a 3-bit binary input into one of eight outputs, with each output corresponding to one of the possible combinations of the input bits. It is a fundamental component in digital systems, used for tasks such as memory addressing, data demultiplexing, and signal routing.

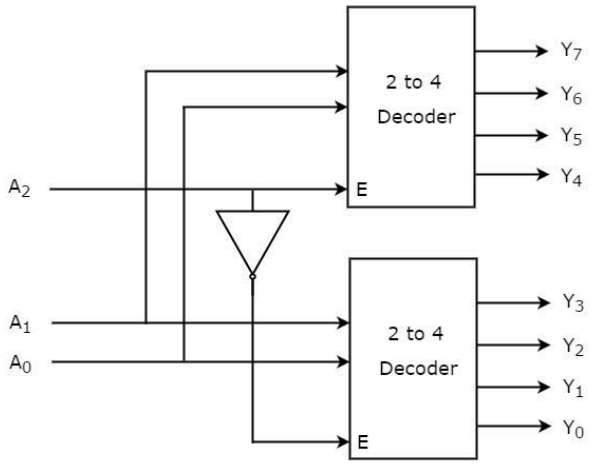

Fig 4.25 : 3 : 8 Decoder Logic Diagram.

Input: I2, I1, I0

Output: Y0, Y1, Y2, Y3, Y4, Y5, Y6, Y7

Truth Table:

Table 4.17: Truth Table of 3 : 8 Decoder.

I2	I1	I0	Y0	Y1	Y2	Y3	Y3	Y3	Y6	Y7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

MGDI 3 : 8 Decoder:

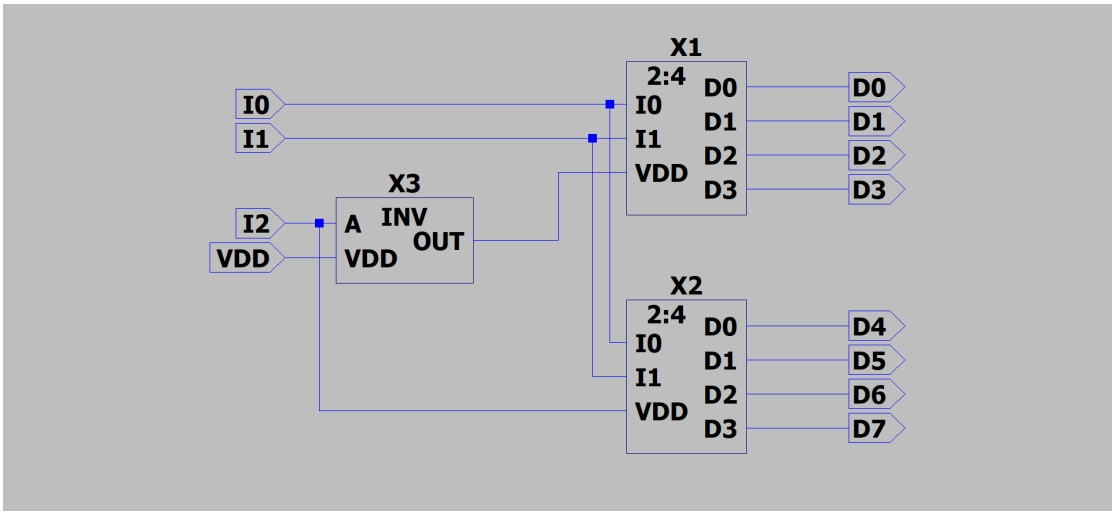

Fig 4.26 : 3 : 8 Decoder Schematic LTSPICE.

To create a 3-to-8 decoder using two 2-to-4 decoders with common inputs A and B, we employ a selective activation method determined by the input combination of A, B, and C. In this configuration, both 2-to-4 decoders receive the same input signals A and B. The enable signal C, however, governs the activation of each decoder: the inverted form of C controls the lower decoder, while the non-inverted signal manages the upper decoder. By leveraging this setup, only one of the decoders is active at any given time, dictated by the state of the enable signal. Consequently, the outputs of the upper decoder serve as the final outputs of the 3-to-8 decoder.

4.4 Design of 1 BIT ALU

In the design of the Arithmetic Logic Unit (ALU), the operation selection is facilitated through a multiplexer (mux) controlled by the OPCODE, serving as the primary means to determine the operation to be executed. The ALU inputs typically include operands A and B, as well as the control signals CIN or BIN, and the OPCODE. The OPCODE, encoded in binary, dictates the operation to be performed by the ALU.

Upon receiving the OPCODE, the ALU utilizes a mux to interpret its binary value and select the appropriate operation. Each unique OPCODE value corresponds to a specific operation, ranging from arithmetic tasks like addition and subtraction to logical functions such as AND, OR, XOR, and XNOR, as well as operations like CLEAR A and NOT A.

Additionally, the mux can also incorporate control signals like CIN or BIN to enable operations like addition with carry or subtraction with borrow. For instance,

during addition (OPCODE 001), the mux may consider CIN to account for carry-in from a previous addition operation. Conversely, during subtraction (OPCODE 100), the mux may utilize BIN to manage borrow-in from the previous subtraction operation. Functioning as the operation selector, the mux directs the appropriate inputs to the ALU's processing units, facilitating the execution of the chosen operation. This dynamic operation selection ensures that the ALU can perform a diverse range of computations based on the provided inputs and the decoded OPCODE.

Designing an Arithmetic Logic Unit (ALU) to perform the specified operations requires careful planning and implementation. Here are the steps to design the ALU:

- **Determine the ALU Inputs:** Identify the input signals required for the ALU operations. In this case, the ALU will require inputs A, B, CIN (carry-in)or BIN(borrow-in), and OPCODE (operation code).
- **Decode the OPCODE:** Use a MUX to interpret the OPCODE and select the appropriate operation to be performed based on its value.
- **Implement the Operations:** For each operation, design the necessary logic circuits to perform the specified function:
    - Operation 000 (CLEAR A): Set A to zero.
    - Operation 001 (ADD): Addition of A, B, and CIN.
    - Operation 010 (AND): Bitwise AND operation between A and B.
    - Operation 011 (OR): Bitwise OR operation between A and B.
    - Operation 100 (SUB): Subtraction of B from A with BIN (borrow-in).
    - Operation 101 (XOR): Bitwise XOR operation between A and B.
    - Operation 110 (XNOR): Bitwise XNOR operation between A and B.
    - Operation 111 (NOT A): Bitwise NOT operation on A.
- **Output Generation:** Generate the ALU output based on the selected operation and the inputs A, B, and CIN or BIN.

Final Design:

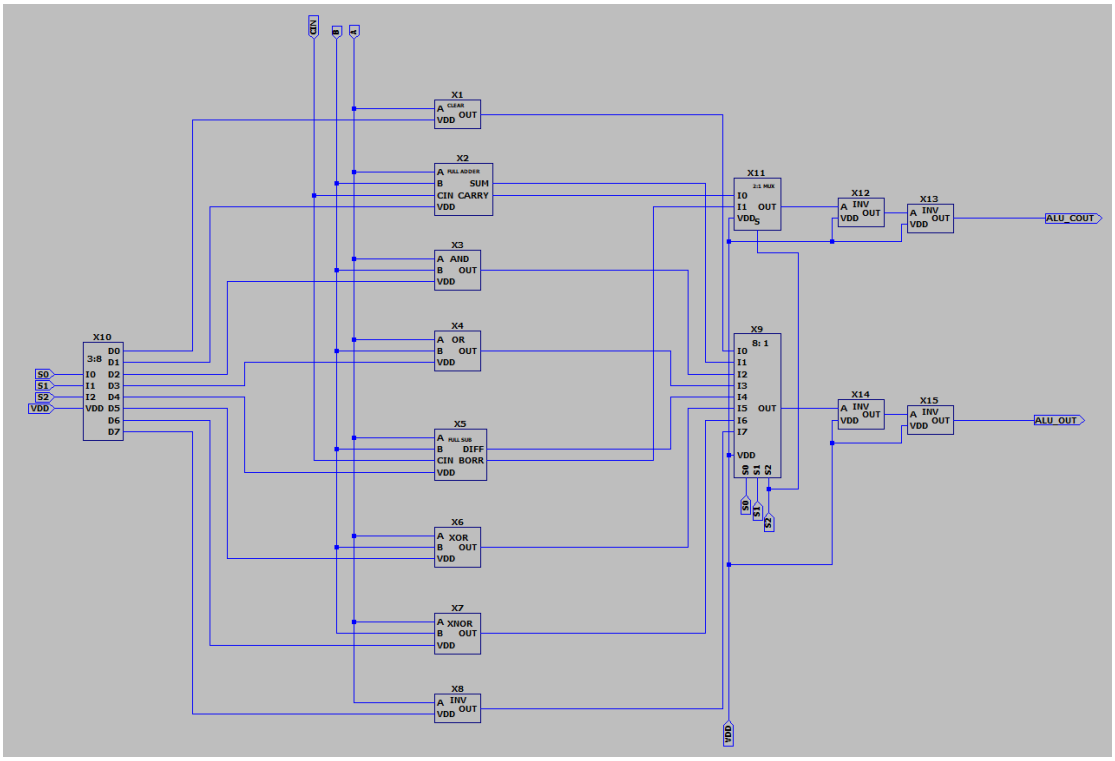

Fig 4.27 : 1 BIT ALU Schematic LTSPICE.

4.5 Design of 2 BIT ALU

To design a 2-bit ALU using 1-bit ALU units in a bit-sliced architecture, we extend the functionality of single-bit ALUs by connecting them in a series, enabling multi-bit operations. Each 1-bit ALU is responsible for handling one bit of the input operands and producing the corresponding bit of the output, along with handling carry/borrow signals between the bits.

Design Steps:

1. **Single-bit ALU Design:** Each 1-bit ALU performs basic operations such as addition, subtraction, AND, OR, XOR, XNOR, NOT, and CLEAR, controlled by the OPCODE. It takes in inputs A, B, a carry-in (CIN), and an OPCODE to produce a single-bit result and a carry-out (COUT).
2. **Connecting 1-bit ALUs:** For a 2-bit ALU, two 1-bit ALUs are connected in series. The least significant bit (LSB) ALU takes the LSBs of the input operands (A0, B0) and the initial carry-in (CIN). The carry-out (COUT) from the LSB ALU is connected to the carry-in (CIN) of the most significant bit (MSB) ALU.

3. **Handling Carry/Borrow:** The carry-out from each 1-bit ALU is crucial for multi-bit operations. In addition and subtraction, the carry-out from the LSB ALU propagates to the MSB ALU, ensuring proper multi-bit arithmetic.
4. **Operation Selection:** The OPCODE is fed to both 1-bit ALUs to ensure synchronized operation across the bits. Each ALU interprets the OPCODE to perform the specified operation on its respective bits of the input operands.

Final Design:

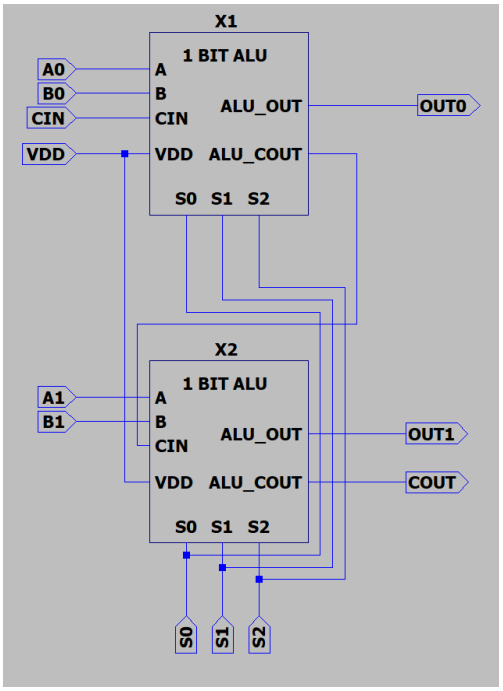

Fig 4.28 : 2 BIT ALU Schematic LTSPICE.

4.6 Design of 4 BIT ALU

To design a 4-bit ALU using two 2-bit ALUs in a bit-sliced architecture, Each 2-bit ALU processes a pair of input bits, and the carry-out from the lower 2-bit ALU is fed into the carry-in of the higher 2-bit ALU, enabling the handling of carry/borrow signals across the full 4-bit operation.

Design Steps:

1. **Design the 2-bit ALUs:** Each 2-bit ALU comprises two 1-bit ALUs that handle two bits of the input operands and produce a 2-bit result along with a carry-out signal.

2. **Connecting the 2-bit ALUs:** For the 4-bit ALU, connect two 2-bit ALUs in series. The first 2-bit ALU (lower) takes the least significant bits (LSBs) of the input operands (A[1:0], B[1:0]) and an initial carry-in (CIN). The second 2-bit ALU (upper) processes the most significant bits (MSBs) of the input operands (A[3:2], B[3:2]) and takes the carry-out from the first 2-bit ALU as its CIN.
3. **Operation Flow:**
    - **Lower 2-bit ALU:** This ALU processes the LSBs (A[1:0] and B[1:0]) and CIN to produce the LSB portion of the result and a carry-out signal (COUT1).
    - **Upper 2-bit ALU:** This ALU takes the MSBs (A[3:2] and B[3:2]) and the carry-out from the lower 2-bit ALU as its carry-in. It processes these inputs to generate result and a final carry-out.

Final Design:

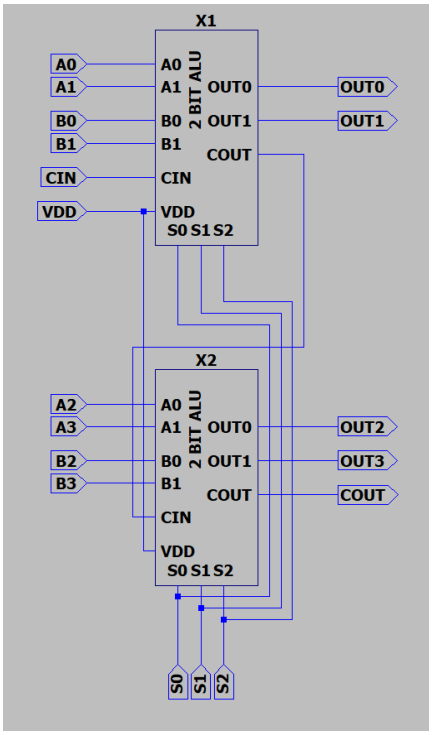

Fig 4.29 : 4 BIT ALU Schematic LTSPICE.

4.7 Design of 8-BIT ALU

To design an 8-bit ALU using two 4-bit ALUs in a bit-sliced architecture, we build upon the principles of connecting smaller ALU units to handle larger bit-widths. Each 4-bit ALU handles four bits of the input operands, and the carry-out from the

lower 4-bit ALU is fed into the carry-in of the higher 4-bit ALU, enabling the handling of carry/borrow signals across the full 8-bit operation.

Design Steps:

1. **Design the 4-bit ALUs:** Each 4-bit ALU is capable of performing basic operations (such as addition, subtraction, AND, OR, XOR, XNOR, NOT, and CLEAR) on 4-bit inputs. It produces a 4-bit result along with a carry-out signal.
2. **Connecting the 4-bit ALUs:** For the 8-bit ALU, connect two 4-bit ALUs in series. The first 4-bit ALU (lower) takes the least significant bits (LSBs) of the input operands (A[3:0], B[3:0]) and an initial carry-in (CIN). The second 4-bit ALU (upper) processes the most significant bits (MSBs) of the input operands (A[7:4], B[7:4]) and takes the carry-out from the first 4-bit ALU as its carry-in (CIN2).
3. **Operation Flow:**
    - **Lower 4-bit ALU:** This ALU processes the LSBs (A[3:0] and B[3:0]) and CIN to produce the lower 4-bit portion of the result and a carry-out signal (COUT1).
    - **Upper 4-bit ALU:** This ALU takes the MSBs (A[7:4] and B[7:4]) and the carry-out from the lower 4-bit ALU (COUT1) as its carry-in (CIN2). It processes these inputs to generate the upper 4-bit portion of the result and a final carry-out signal (COUT2).
4. **Handling OPCODE:** Both 4-bit ALUs receive the same OPCODE, which dictates the operation to be performed. This ensures that both ALUs execute the same arithmetic or logical function on their respective input bits.

The 8-bit ALU uses two 4-bit ALUs, and the lower 4-bit ALU handles the least significant 4 bits of the input operands and produces a partial result and a carry-out. This carry-out is then fed into the carry-in of the upper 4-bit ALU, which processes the most significant 4 bits of the input operands and generates the final 4-bit result along with the final carry-out. By connecting the two 4-bit ALUs in series and managing the carry/borrow propagation, the 8-bit ALU can efficiently perform complex operations with minimal hardware complexity.

Final Design:

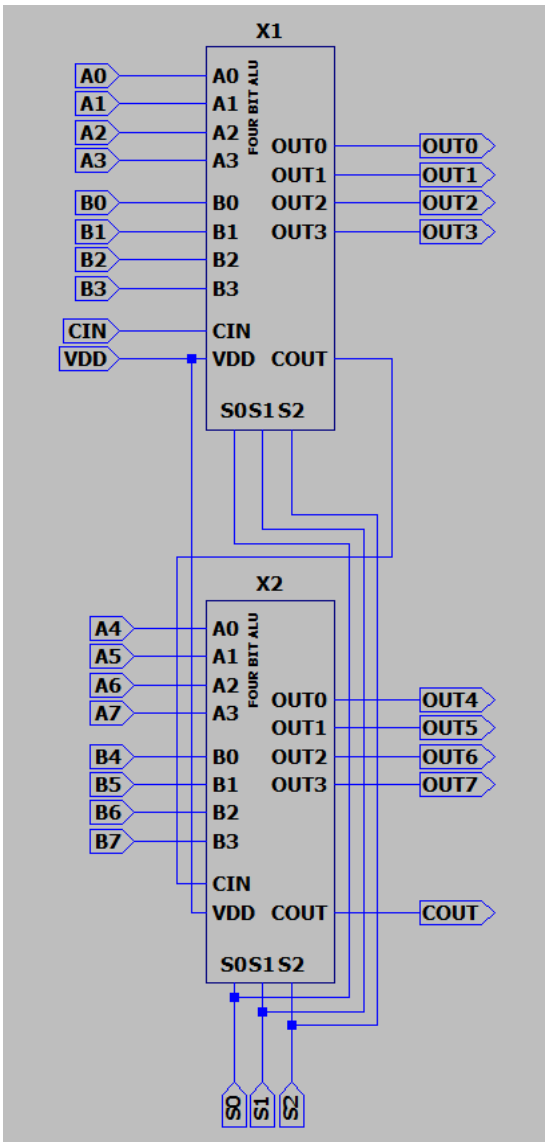

Fig 4.30 : 8 BIT ALU Schematic LTSPICE.

Chapter 5

LTSpice

5.1 Introduction to LTSpice

LTspice is a high-performance SPICE (Simulation Program with Integrated Circuit Emphasis) simulator developed by Linear Technology, now a part of Analog Devices. It is widely recognized for its speed, accuracy, and comprehensive set of features that cater to the needs of electronic design engineers. LTspice allows for the simulation and analysis of analog and digital circuits, making it an indispensable tool in the field of electronics.

5.2 Key Features:

Simulation Capabilities:

- Analog and Digital Simulation: LTspice excels in simulating both analog and digital circuits, offering designers the flexibility to work on a variety of projects. Analog simulation encompasses circuits like amplifiers, filters, oscillators, and power supplies, while digital simulation includes logic gates, flip-flops, counters, and complex digital systems.
- Transient Analysis: Transient analysis in LTspice allows users to observe the time-domain behavior of circuits. This type of analysis is crucial for understanding how voltages and currents change over time in response to various inputs and stimuli. It helps in designing circuits that need to respond to time-varying signals, such as pulse circuits and the transient response of amplifiers.
- AC Analysis: AC analysis provides insight into the frequency response of circuits. It is essential for designing and analyzing circuits where the behavior varies with frequency, such as filters, amplifiers, and oscillators. LTspice can generate Bode plots, phase plots, and other frequency-dependent characteristics.
- DC Analysis: DC analysis determines the operating point of a circuit by calculating the DC voltages and currents at all nodes. This analysis is fundamental for biasing circuits, ensuring that transistors and other active devices operate within their desired regions.

- Noise Analysis: Noise analysis in LTspice evaluates the noise performance of circuits, which is critical for high-precision and low-noise designs. It helps in identifying the sources and levels of noise, enabling designers to optimize their circuits for better performance.

Component Libraries:

- LTspice includes an extensive library of components that cover a wide range of electronic devices. The library includes:
- Passive Components: Resistors, capacitors, inductors, and transformers are fundamental elements available in the library. Users can define specific parameters such as resistance, capacitance, inductance, and coupling coefficients.
- Semiconductor Devices: The library contains models for various semiconductor devices, including diodes, BJTs, JFETs, and MOSFETs. These models are essential for simulating the behavior of transistors in analog and digital circuits.
- Integrated Circuits: A range of integrated circuits such as operational amplifiers, voltage regulators, and special function ICs are included. These components come with predefined models that replicate the actual behavior of their real-world counterparts.
- Power Electronics: LTspice supports power electronic components such as IGBTs, power MOSFETs, and diodes. It also includes models for magnetic components like inductors and transformers, which are crucial for simulating power supply circuits.
- User-Defined Models: Users can create and import custom models for specific components that are not available in the standard library. This flexibility allows for the inclusion of proprietary or specialized parts in simulations.

User Interface:

- Schematic Capture: LTspice provides an intuitive graphical interface for drawing circuit schematics. Users can easily place components on the schematic, connect them with wires, and configure their properties. The drag-and-drop interface and extensive component library simplify the process of circuit design.

- Waveform Viewer: The waveform viewer is a powerful tool for visualizing simulation results. It allows users to plot multiple waveforms, measure parameters such as voltage, current, and power, and perform detailed analysis. The viewer supports various types of plots, including time-domain, frequency-domain, and noise analysis plots.
- Parameter Stepping and Monte Carlo Analysis: LTspice supports parameter stepping, which allows users to vary component values or operating conditions across a specified range and observe the effects on circuit behavior. Monte Carlo analysis enables statistical analysis of circuits, accounting for variations in component values due to manufacturing tolerances.

Performance and Optimization

LTspice's comprehensive library of analog and digital components, coupled with its intuitive user interface, empowers engineers to simulate a wide range of circuits with ease. Additionally, its seamless integration with industry-standard SPICE models enables precise modeling of real-world components, ensuring simulation results closely match physical measurements.

Key performance features include:

1. Multi-Threading: LTspice leverages the multi-threading capabilities of modern CPUs to accelerate simulation tasks. This allows for faster processing of simulations, particularly for complex and large-scale circuits.
2. Optimized Algorithms: The software uses highly optimized algorithms for solving circuit equations, which enhances the speed and accuracy of simulations. This is particularly beneficial for transient and AC analyses where computational efficiency is critical.
3. Hierarchical Design: LTspice supports hierarchical design, enabling users to break down complex circuits into manageable sub-circuits. This modular approach simplifies the design process and improves simulation performance by allowing parallel processing of sub-circuits.

Applications

LTspice is used in a wide range of applications, including:

- Analog Circuit Design: Designing and analyzing amplifiers, filters, oscillators, and power supplies. LTspice provides detailed insights into the behavior of analog circuits, allowing for optimization and validation.
- Digital Circuit Design: Simulating logic gates, flip-flops, counters, and complex digital systems. LTspice supports digital simulation, enabling the design and testing of digital circuits and systems.
- Mixed-Signal Design: Combining analog and digital components to create mixed-signal circuits, such as ADCs, DACs, and signal conditioning circuits. LTspice can handle the interactions between analog and digital parts of the circuit.
- Power Electronics: Simulating switch-mode power supplies, motor drivers, and inverters. LTspice's extensive library of power electronics components and its ability to handle high-current and high-voltage circuits make it ideal for power electronics design.
- RF Design: Designing and analyzing RF amplifiers, mixers, and communication circuits. LTspice supports high-frequency simulations, enabling the design of RF and microwave circuits.

5.3 Overview

LTspice is an indispensable tool for electronic circuit simulation, offering a comprehensive suite of features that cater to both simple and highly complex circuit designs. Its combination of speed, accuracy, and user-friendly interface, along with extensive component libraries and advanced analysis capabilities, makes it a preferred choice among engineers, educators, and researchers for designing, testing, and optimizing electronic circuits.

Chapter 6

VERIFICATION AND VALIDATION OF CIRCUIT DESIGNS USING LTSPICE

6.1 Introduction

The basic gates and combinational logic circuits designed using the Modified Gate Diffusion Input (MGDI) technique were rigorously tested using LTspice to ensure their functionality and performance. The simulation environment in LTspice provided an accurate representation of real-world behavior, allowing for comprehensive analysis. Each basic gate, including AND, OR, NOT, NAND, NOR, XOR, and XNOR, was subjected to a series of input combinations to verify their truth tables. The outputs were monitored to confirm that they matched the expected logical outcomes. Similarly, combinational logic circuits such as the half adder, full adder, half subtractor, full subtractor, multiplexers (2:1, 4:1, and 8:1), and decoders (2:4 and 3:8) were simulated. The correct propagation of signals through these circuits was verified by applying various input scenarios and observing the outputs.

Now, we will be presenting the detailed testing process and results for the 8-bit Arithmetic Logic Unit (ALU) designed using these MGDI-based gates and combinational logic. The 8-bit ALU's performance was validated by simulating a comprehensive set of operations, including addition, subtraction, logical operations (AND, OR, XOR, XNOR, NOT), and clear functions, each corresponding to specific opcodes. The outputs were thoroughly checked against expected results to ensure that the ALU operates correctly across all supported operations. This focused testing on the 8-bit ALU highlights the reliability of the MGDI technique in complex digital circuit design. Note that all the simulations performed were done with respect to PTM files which is mentioned in Appendix - 1

6.2 Test Setup for 8-bit ALU in LTSpice:

Testing 8-bit ALU in LTSpice, follow these steps to create a test environment that will generate appropriate inputs and verify the outputs for various operations:

- **Input Signal Generation:**
    - Define input combinations for A and B, with 0V representing logic 0 and 1V representing logic 1.

    - Include signals for CIN or BIN, which are also binary and switch between 0V and 1V.
- **Control Signal Setup:**
    - Create control signals to select the operations (e.g., CLR_A, ADD, AND, OR, SUB, XOR, XNOR, NOT_A).
    - Use PWL sources to change the control signals at specific times to test each operation sequentially.
- **Simulation Sequence:**
    - Set up a sequence of input combinations and operations to run automatically during the simulation.
    - Ensure the sequence covers all combinations of inputs A and B along with all operations.
- **Running the Simulation:**
    - Use the .tran (transient analysis) command to define the duration and time step of the simulation.
    - Run the simulation to observe the output waveforms.
- **Output Verification:**
    - Use the waveform viewer in LTSpice to inspect the outputs (Result[7:0], COUT, BO) for each test case.
    - Compare the observed outputs with the expected results to verify correct functionality.

Following the detailed analysis of each test case, a photo is attached for every scenario as proof of verification, authenticity, and genuineness. These photos serve as visual evidence of the results obtained during testing, providing transparency and reinforcing the accuracy of the ALU's performance. By including these images, we ensure that each step of the testing process is well-documented and verifiable, allowing for easy identification and resolution of any discrepancies. This comprehensive approach not only enhances confidence in the correctness of the ALU's functionality but also ensures that the testing process is thorough and reliable.

6.2.1 Test Case 1

In this test case, we aim to evaluate the functionality of an 8-bit Arithmetic Logic Unit (ALU) with specific input conditions. The inputs for this scenario are set

as follows: the first operand A is represented by the binary value 0000_1101, which corresponds to the decimal value 13. The second operand B is 0000_0111, equivalent to the decimal value 7. Additionally, the carry-in (CIN) is set to 0, indicating that no initial carry is introduced into the arithmetic operations.

Input: A = 0000_1101 (13), B = 0000_0111 (7), CIN/BIN = 0

Table 5.1: Test Case 1 Results.

Operation	Obtained Results	Obtained COUT	Expected Results	Expected COUT
CLEAR A	00000000	0	00000000	0
ADD	00010100	0	00010100	0
AND	00000101	0	00000101	0
OR	00001111	0	00001111	0
SUB	00000101	0	00000101	0
XOR	00001010	0	00001010	0
XNOR	11110101	0	11110101	0
NOT A	11110010	0	11110010	0

Proof: The output obtained after execution of TC1.

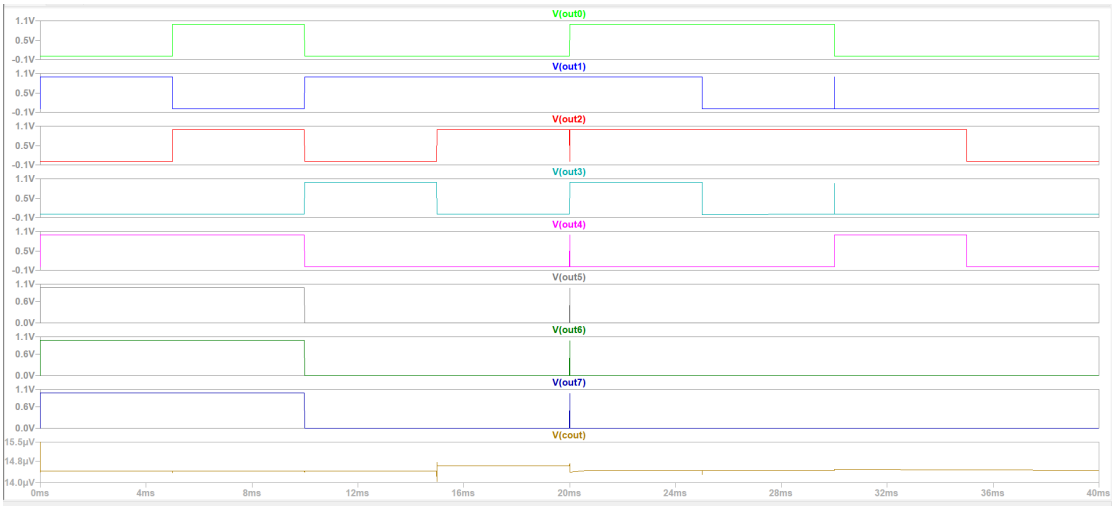

Fig 5.1: Waveforms of Test Case 1.

6.2.2 Test Case 2

In this test case, we continue to evaluate the functionality of the 8-bit Arithmetic Logic Unit (ALU) under different input conditions. The inputs for this scenario are as follows: the first operand A is represented by the binary value 1111_1111(255). The second operand B is 0000_0001(1) and CIN is set to 1.

Input: A = 1111_1111 (255), B = 0000_0001 (1), CIN/BIN = 1

Table 5.2: Test Case 2 Results.

Operation	Obtained Results	Obtained COUT	Expected Results	Expected COUT
CLEAR A	00000000	0	00000000	0
ADD	00000001	1	00000001	1
AND	00000001	0	00000001	0
OR	11111111	0	11111111	0
SUB	11111101	0	11111101	0
XOR	11111110	0	11111110	0
XNOR	00000001	0	00000001	0
NOT A	00000000	0	00000000	0

Proof: The output obtained after execution of TC2.

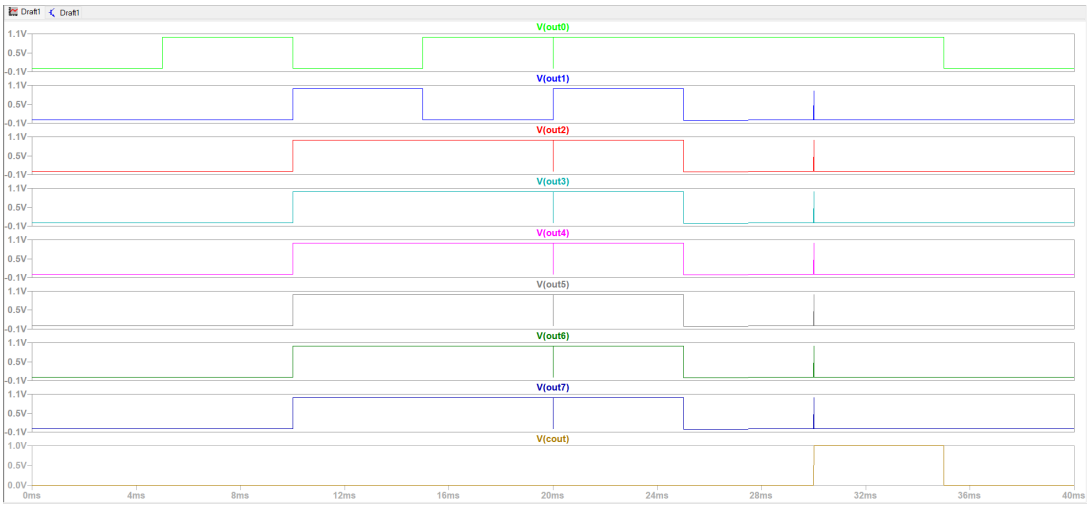

Fig 5.2: Waveforms of Test Case 2.

6.2.3 Test Case 3

In this test case, we aim to assess the performance of the 8-bit Arithmetic Logic Unit (ALU) with specific input parameters. The first operand A is represented by the binary value 1100_1100, corresponding to the decimal value 204. The second operand B is 1010_1010, equivalent to the decimal value 170. The CIN is set to 0.

Input: A = 1100_1100 (204), B = 1010_1010 (170), CIN/BIN = 0

Table 5.3: Test Case 3 Results.

Operation	Obtained Results	Obtained COUT	Expected Results	Expected COUT
CLEAR A	00000000	1	00000000	0
ADD	01110110	1	01110110	1
AND	10001000	1	10001000	0
OR	11101110	1	11101110	0
SUB	00100010	0	00100010	0
XOR	01100110	0	01100110	0
XNOR	10011001	0	10011001	0
NOT A	00110011	0	00110011	0

Proof: The output obtained after execution of TC3.

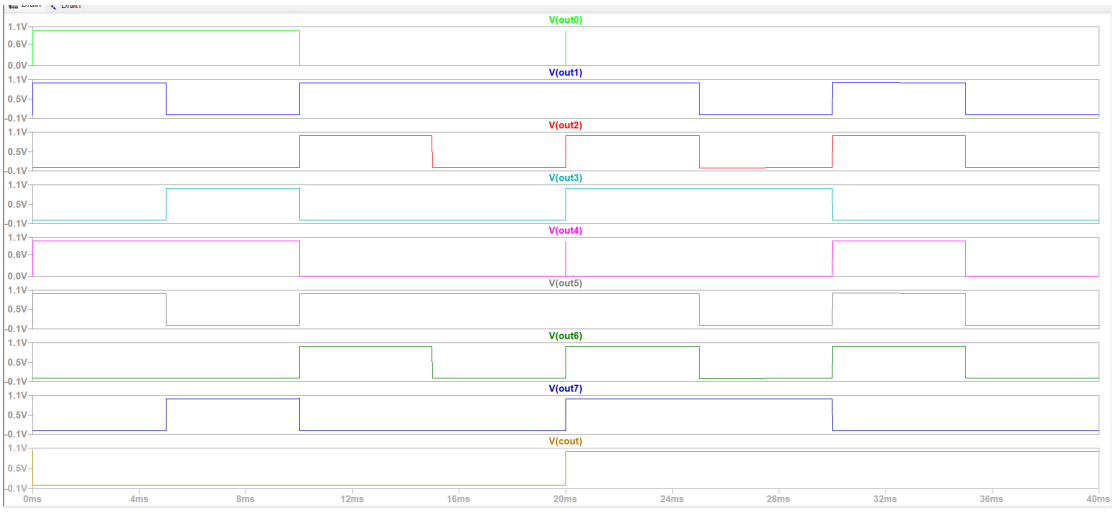

Fig 5.3: Waveforms of Test Case 3.

6.2.4 Test Case 4

In Test Case 4, we aim to evaluate the performance of the 8-bit Arithmetic Logic Unit (ALU) under specific input conditions. The first operand A is represented by the binary value 0000_0000, which corresponds to the decimal value 0. The second operand B is 1111_1111, equivalent to the decimal value 255.

Input: A = 0000_0000 (0), B = 1111_1111 (255), CIN/BIN = 0

Table 5.4: Test Case 4 Results.

Operation	Obtained Results	Obtained COUT	Expected Results	Expected COUT
CLEAR A	00000000	0	00000000	0
ADD	11111111	0	11111111	0
AND	00000000	0	00000000	0
OR	11111111	0	11111111	0
SUB	00000001	1	00000001	1
XOR	11111111	0	11111111	0
XNOR	00000000	0	00000000	0
NOT A	11111111	0	11111111	0

Proof: The output obtained after execution of TC4.

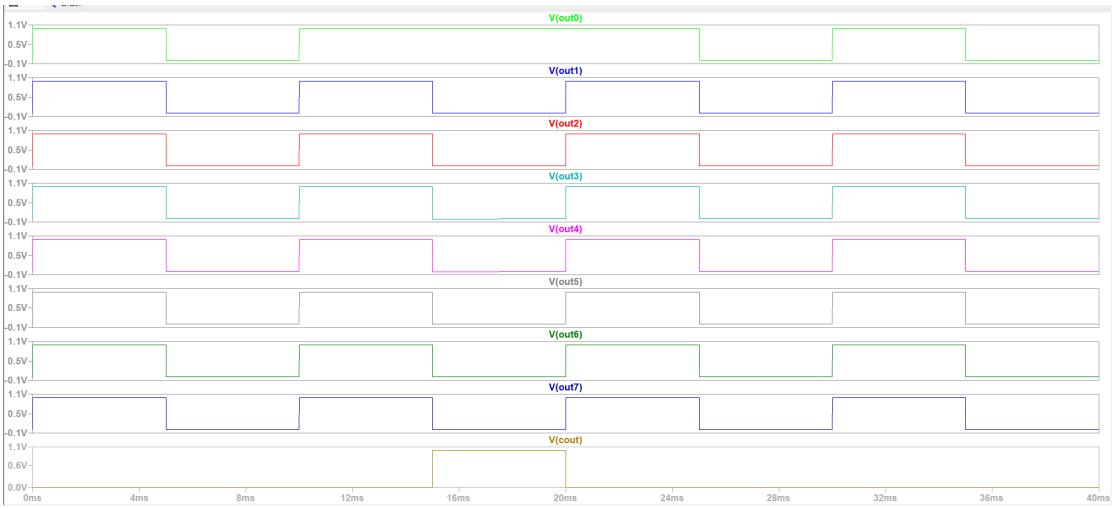

Fig 5.4: Waveforms of Test Case 4.

6.3 Conclusion

Upon thorough analysis of the test results, it can be concluded that the 8-bit Arithmetic Logic Unit (ALU) has demonstrated correct functionality across all tested operations and input conditions. The outputs produced by the ALU for Test Cases 1, 2, 3, and 4 align precisely with the expected results, validating the accuracy and reliability of its arithmetic and logical operations. These results provide confidence in the ALU's ability to handle diverse input scenarios effectively, ranging from small to large operand values and with varying initial carry conditions. With these positive outcomes, it can be affirmed that the ALU meets the specified requirements and can be deemed suitable for integration into broader digital circuits and systems.

Chapter 7

RESULTS

7.1 Area Consumption

Area consumption is a critical factor in digital circuit design, directly impacting the cost, performance, and scalability of integrated circuits (ICs). The area occupied by a circuit on a semiconductor die is directly proportional to the number of transistors used, influencing the overall chip size.

7.1.2 Transistor Count

The table provides a detailed comparison of the transistor counts for both NMOS and PMOS transistors in traditional CMOS logic versus MGDI technology.

Table 7.1 : Transistor Count MGDI vs CMOS.

LOGIC	MGDI	CMOS
AND	2	6
OR	2	6
NAND	4	4
NOR	4	4
XOR	4	12
XNOR	4	12
NOT	2	2
HALF ADDER	6	18
FULL ADDER	14	42
HALF SUBTRACTOR	8	20
FULL SUBTRACTOR	18	46
2 : 1 MUX	2	20

LOGIC	MGDI	CMOS
4 : 1 MUX	6	60
8 : 1 MUX	14	140
2 : 4 DECODER	12	28
3 : 8 DECODER	26	58
1 BIT ALU	98	350
8 BIT ALU	784	2800

The adoption of Modified Gate Diffusion Input (MGDI) technology has demonstrated a significant reduction in transistor count, achieving a remarkable 72 percent decrease compared to traditional Complementary Metal-Oxide-Semiconductor (CMOS) logic. This substantial reduction highlights the area efficiency of MGDI, as fewer transistors directly translate to smaller silicon area requirements for the same logic functions. By utilizing innovative circuit configurations and minimizing redundant transistor usage, MGDI not only conserves valuable chip real estate but also enhances overall design compactness. This efficiency is particularly advantageous for high-density applications, where space constraints are critical.

7.2 Power Consumption Analysis of the Arithmetic Logic Unit (ALU)

The power consumption analysis of the Arithmetic Logic Unit (ALU) reveals an average power usage of 91.463 µW. This average value provides a representative measure of the ALU's power efficiency during typical operations, serving as a crucial metric for optimizing performance and energy usage. Additionally, the power consumption fluctuates within a range, with a minimum power draw of 4.8566 µW and a maximum of 178.08 µW. These values highlight the variability in power requirements based on different operational states and workloads of the ALU. Understanding this power consumption spectrum is essential for designing energy-efficient systems, ensuring that the ALU operates within acceptable power limits while maintaining optimal performance.

Minimum Power: 4.8566 µW

Maximum Power: 178.08 µW

Average Power: 91.463 µW

Proof:

- The maximum power consumed by the 8 BIT ALU.

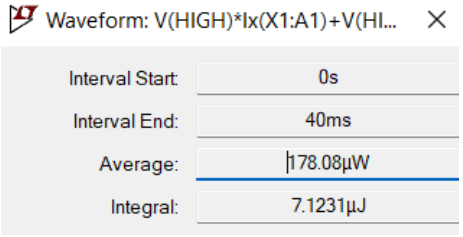

Fig 7.1 : Maximum Power Consumed by MGDI 8 BIT ALU.

- The minimum power consumed by the 8 BIT ALU

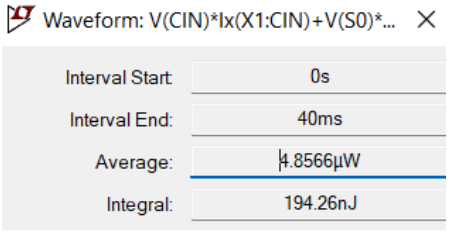

Fig 7.2 : Minimum Power Consumed by MGDI 8 BIT ALU.

Chapter 8

CONCLUSION AND FUTURE SCOPE

8.1 Conclusion

MGDI technology has demonstrated a significant 72 percent reduction in transistor count compared to traditional CMOS logic. This dramatic reduction in transistor usage translates directly to a smaller silicon area, making MGDI an exceptionally area-efficient alternative for digital circuit design.

MGDI achieves this efficiency through innovative circuit configurations that minimize redundant transistor usage. By connecting the gate inputs differently and often sharing diffusion regions, MGDI can implement logic functions with far fewer transistors. For instance, an AND gate in MGDI can be realized with just two transistors, compared to six in CMOS, resulting in substantial area savings.

The practical application of MGDI is exemplified in the design of an 8-bit Arithmetic Logic Unit (ALU) using cascaded 4-bit ALUs. This modular approach optimizes carry propagation and enhances computational efficiency, simplifying scalability and implementation. The efficient design methodology of MGDI, combined with the strategic use of 4-bit ALU blocks, showcases the potential for achieving high-performance computing with reduced hardware complexity.

Moreover, the power consumption analysis of the ALU indicates an average power usage of 91.463 μW, with values ranging from a minimum of 4.8566 μW to a maximum of 178.08 μW. These metrics highlight the ALU's balanced power efficiency during operation, essential for optimizing performance while maintaining energy efficiency. This efficiency makes MGDI a highly advantageous choice for designing cost-effective, high-performance, and scalable digital systems. The insights gained from this analysis underscore the potential of MGDI to revolutionize modern electronics and integrated circuit design, providing a pathway for more compact and efficient digital solutions.

8.2 Future Scope

The future scope of Modified Gate Diffusion Input (MGDI) technology in digital circuit design is vast and promising. As the demand for smaller, faster, and more energy-efficient electronic devices grows, MGDI is poised to play a crucial role in meeting these needs. The integration of MGDI into advanced semiconductor nodes, such as 5nm and beyond, could further enhance its area and power efficiency benefits,

leading to even more compact and energy-efficient circuits. Its suitability for low-power applications makes it ideal for wearables, IoT devices, and battery-operated systems, where ultra-low-power operation is critical. MGDI's efficiency in managing complex logic functions with fewer transistors can significantly benefit high-performance computing, providing higher computational speeds with lower power consumption.

REFERENCES

[1] Morgenstein A. Fish, I.A. Wagner. 2002. GateDiffusion Input (GDI) - A Power Efficient Method for Digital Combinational Circuits. IEEE Trans. VLSI. 10(5): 566-581.

[2] Pinninti Kishore P.V. Sridevi, K. Babulu. 2016. Low Power and Optimized Ripple Carry Adder and Carry Select Adder Using MOD-GDI Technique. Proceedings of Microelectronics, Electromagnetics and Telecommunications. Lecture Notes in Electrical Engineering, Springer India. pp. 159-171.

[3] Kiat -Seng Yeo, Kaushik Roy. 2009. Low-Voltage, Low-Power VLSI Subsystems. Tata McGraw-Hill edition. pp. 83-85.

[4] T. Kim, W. Jao, and S.Tjiang, 1998. Circuit optimization using carry saver adder cells. IEEE Trans. Computer-aided design of integrated circuits and systems. 17(10): 974-984.

[5] Pinninti Kishore, P.V. Sridevi, K. Babulu, K.S. Pradeep Chandra. 2015. A Novel Low Power and Area Efficient Carry-Lookahead Adder using ModGDI Technique. International Journal of Scientific and Research. 4(5): 1205-1210.

[6] S. Hanson, B. Zhai, K. Bernstein, D. Blaauw, A. Bryant, L. Chang, K. K. Das, W. Haensch, E. J. Nowak and D. Sylvester, 2006. Ultralow-voltage, minimum-energy CMOS. IBM Journal of Research and Development. 50(4-5): 469-490.

[7]. Aiyyappu Surendrababu N.Ashokkumar. “Design of ALU using 2T XOR Gate and Decoder”, International Journal on Cybernetics & Informatics (IJCI) Vol. 10, No.3, June 2023.

[8]. Anil Nageshwar Rangapure, Dr.Kiran. “8-bit Arithmetic Logic Unit Design using Modified Gate Diffusion Input (m-GDI) Technique” .International Journal of Advances in Engineering and Management (IJAEM), Volume 3, Issue 9 Sep 2021, ISSN: 2395-5252.

[9]. 8-bit Arithmetic Logic Unit. Samuel Winchenbach, Mohammed Driss, University of Maine, Orono, 2019.

[10]. P. Sai Krishna, P. Brundavani. “Design and Implementation of Low Power 16-Bit ALU using MGDI Technique”, ISSN 2322-0929, Vol.05, Issue.10, October-2019, Pages:0954-0959.

[11]. Harshmaniyadav Uday Panwar. “Design of 8-Bit ALU Design using GDI Techniques with Less Power and Delay”. International Journal of Recent Technology and Engineering (IJRTE), Volume-8 Issue-4, November 2017. ISSN: 2277-3878.

[12]. Dr. K. Srinivasulu, M. Shiva Kumar, Chetempally Sridhar Goud. “Design of a Low Power Area Efficient ALU Using Modified GDI Multiplexer.” October 3 2017.

[13]. N.Srinu, M.Kedhar, O.Ajay, “Design of Low-Power ALU Using GDI Technique.” 2018 IJCRT, Volume 6, Issue 2 April 2016, ISSN: 2320-2882.

[14]. Anitesh Sharma, Ravi Tiwari. “Low Power 8-Bit ALU Design Using Full Adder and Multiplexer”. International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET). IEEE 2016.

[15]. A.S. Prabhu, B. Naveena, K. Parimaladevi, M. Samundeswari, P. Thilagavathy. “Serial Divider Using Modified GDI Technique.” International Journal of Innovative Research in Electrical (IJIRE). Vol. 3, Issue 10, October 2016.

[16]. Pinninti Kishore, K. Babulu and P. V. Sridevi. “Low Power and High-Speed Carry Save Adder Using Modified Gate Diffusion Input Technique.” ARPN Journal of Engineering and Applied Sciences. VOL. 11, NO. 21, NOVEMBER 2016, ISSN 1819-6608.

APPENDIX - I

SIMULATION PARAMETERS

The project simulations were performed using Predictive Technology Model (PTM) files. PTM files accurately emulate the behavior of real-world transistors, ensuring precise representation of electrical characteristics. This methodology enhances the reliability and applicability of the project's findings to practical scenarios.

Technology Node: 65nm.

PMOS PTM FILE - 65nm

```
*Customized PTM 65nm PMOS
.model  pmos  pmos  level = 54
+version = 4.0 binunit = 1      paramchk= 1   mobmod  = 0
+capmod  = 2            igcmod  = 1    igbmod  = 1    geomod  = 1
+diomod  = 1  rdsmod  = 0    rbodymod= 1  rgatemod= 1
+permod  = 1  acnqsmod= 0  trnqsmod= 0

* parameters related to the technology node
+tnom = 27     epsrox = 3.9
+eta0 = 0.0058          nfactor = 1.9   wint = 5e-09
+cgso = 1.5e-10         cgdo = 1.5e-10          xl = -3e-08

* parameters customized by the user
+toxe = 1.95e-09        toxp = 1.2e-09 toxm = 1.95e-09        toxref = 1.95e-09
+dtox = 7.5e-10         lint = 5.25e-09
+vth0 = -0.378k1 = 0.456      u0 = 0.00548   vsat = 70000
+rdsw = 165    ndep = 1.97e+18        xj = 1.96e-08

*secondary parameters
+ll     = 0             wl      = 0             lln     = 1             wln     = 1
+lw     = 0             ww      = 0             lwn     = 1             wwn     = 1
+lwl    = 0             wwl     = 0             xpart   = 0
+k2     = -0.01         k3      = 0
+k3b    = 0             w0      = 2.5e-006      dvt0    = 1             dvt1    = 2
```

+dvt2 = -0.032 dvt0w = 0 dvt1w = 0 dvt2w = 0

+dsub = 0.1 minv = 0.05 voffl = 0 dvtp0 = 1e-009

+dvtp1 = 0.05 lpe0 = 0 lpeb = 0

+ngate = 2e+020 nsd = 2e+020 phin = 0

+cdsc = 0.000 cdscb = 0 cdscd = 0 cit = 0

+voff = -0.126 etab = 0

+vfb = 0.55 ua = 2.0e-009 ub = 0.5e-018

+uc = 0 a0 = 1.0 ags = 1e-020

+a1 = 0 a2 = 1 b0 = -1e-020 b1 = 0

+keta = -0.047 dwg = 0 dwb = 0 pclm = 0.12

+pdiblc1 = 0.001 pdiblc2 = 0.001 pdiblcb = 3.4e-008 drout = 0.56

+pvag = 1e-020 delta = 0.01 pscbe1 = 8.14e+008 pscbe2 = 9.58e-007

+fprout = 0.2 pdits = 0.08 pditsd = 0.23 pditsl = 2.3e+006

+rsh = 5 rsw = 85 rdw = 85

+rdswmin = 0 rdwmin = 0 rswmin = 0 prwg = 3.22e-008

+prwb = 6.8e-011 wr = 1 alpha0 = 0.074 alpha1 = 0.005

+beta0 = 30 agidl = 0.0002 bgidl = 2.1e+009 cgidl = 0.0002

+egidl = 0.8

+aigbacc = 0.012 bigbacc = 0.0028 cigbacc = 0.002

+nigbacc = 1 aigbinv = 0.014 bigbinv = 0.004 cigbinv = 0.004

+eigbinv = 1.1 nigbinv = 3 aigc = 0.69 bigc = 0.0012

+cigc = 0.0008 aigsd = 0.0087 bigsd = 0.0012 cigsd = 0.0008

+nigc = 1 poxedge = 1 pigcd = 1 ntox = 1

+xrcrg1 = 12 xrcrg2 = 5

+cgbo = 2.56e-011 cgdl = 2.653e-10

+cgsl = 2.653e-10 ckappas = 0.03 ckappad = 0.03 acde = 1

+moin = 15 noff = 0.9 voffcv = 0.02

+kt1 = -0.11 kt1l = 0 kt2 = 0.022 ute = -1.5

+ua1 = 4.31e-009 ub1 = 7.61e-018 uc1 = -5.6e-011 prt = 0

+at = 33000

+fnoimod = 1 tnoimod = 0

+jss = 0.0001 jsws = 1e-011 jswgs = 1e-010 njs = 1

+ijthsfwd= 0.01 ijthsrev= 0.001 bvs = 10 xjbvs = 1

+jsd = 0.0001 jswd = 1e-011 jswgd = 1e-010 njd = 1

+ijthdfwd= 0.01 ijthdrev= 0.001 bvd = 10 xjbvd = 1

+pbs = 1 cjs = 0.0005 mjs = 0.5 pbsws = 1

+cjsws = 5e-010 mjsws = 0.33 pbswgs = 1 cjswgs = 3e-010

+mjswgs = 0.33 pbd = 1 cjd = 0.0005 mjd = 0.5

+pbswd = 1 cjswd = 5e-010 mjswd = 0.33 pbswgd = 1

+cjswgd = 5e-010 mjswgd = 0.33 tpb = 0.005 tcj = 0.001

+tpbsw = 0.005 tcjsw = 0.001 tpbswg = 0.005 tcjswg = 0.001

+xtis = 3 xtid = 3

+dmcg = 0e-006 dmci = 0e-006 dmdg = 0e-006 dmcgt = 0e-007

+dwj = 0.0e-008 xgw = 0e-007 xgl = 0e-008

+rshg = 0.4 gbmin = 1e-010 rbpb = 5 rbpd = 15

+rbps = 15 rbdb = 15 rbsb = 15 ngcon = 1

NMOS PTM FILE - 65nm

```
* Customized PTM 65nm NMOS
.model  nmos  nmos  level = 54
+version = 4.0 binunit = 1      paramchk= 1   mobmod  = 0
+capmod  = 2           igcmod  = 1    igbmod  = 1    geomod  = 1
+diomod  = 1  rdsmod  = 0    rbodymod= 1  rgatemod= 1
+permod  = 1  acnqsmod= 0  trnqsmod= 0

* parameters related to the technology node
+tnom = 27     epsrox = 3.9
+eta0 = 0.0058          nfactor = 1.9    wint = 5e-09
+cgso = 1.5e-10         cgdo = 1.5e-10          xl = -3e-08

* parameters customized by the user
+toxe = 1.85e-09        toxp = 1.2e-09 toxm = 1.85e-09        toxref = 1.85e-09
+dtox = 6.5e-10         lint = 5.25e-09
+vth0 = 0.429  k1 = 0.497      u0 = 0.04861   vsat = 124340
+rdsw = 165    ndep = 2.6e+18          xj = 1.96e-08

* secondary parameters
+ll      = 0            wl      = 0            lln     = 1            wln     = 1
+lw      = 0            ww      = 0            lwn     = 1            wwn     = 1
+lwl     = 0            wwl     = 0            xpart   = 0
+k2      = 0.01         k3      = 0
+k3b     = 0            w0      = 2.5e-006     dvt0    = 1            dvt1    = 2
+dvt2    = -0.032       dvt0w   = 0            dvt1w   = 0            dvt2w   = 0
+dsub    = 0.1          minv    = 0.05         voffl   = 0      dvtp0  = 1.0e-009
+dvtp1   = 0.1          lpe0    = 0            lpeb    = 0
+ngate   = 2e+020       nsd     = 2e+020       phin    = 0
+cdsc    = 0.000        cdscb   = 0            cdscd   = 0            cit     = 0
+voff    = -0.13        etab    = 0
+vfb     = -0.55        ua      = 6e-010       ub      = 1.2e-018
+uc      = 0            a0      = 1.0          ags     = 1e-020
+a1      = 0            a2      = 1.0          b0      = 0            b1      = 0
```

+keta = 0.04 dwg = 0 dwb = 0 pclm = 0.04

+pdiblc1 = 0.001 pdiblc2 = 0.001 pdiblcb = -0.005 drout = 0.5
+pvag = 1e-020 delta = 0.01 pscbe1 = 8.14e+008 pscbe2 = 1e-007

+fprout = 0.2 pdits = 0.08 pditsd = 0.23 pditsl = 2.3e+006
+rsh = 5 rsw = 85 rdw = 85
+rdswmin = 0 rdwmin = 0 rswmin = 0 prwg = 0
+prwb = 6.8e-011 wr = 1 alpha0 = 0.074 alpha1 = 0.005

+beta0 = 30 agidl = 0.0002 bgidl = 2.1e+009 cgidl = 0.0002

+egidl = 0.8

+aigbacc = 0.012 bigbacc = 0.0028 cigbacc = 0.002
+nigbacc = 1 aigbinv = 0.014 bigbinv = 0.004 cigbinv = 0.004

+eigbinv = 1.1 nigbinv = 3 aigc = 0.012 bigc = 0.0028

+cigc = 0.002 aigsd = 0.012 bigsd = 0.0028 cigsd = 0.002
+nigc = 1 poxedge = 1 pigcd = 1 ntox = 1

+xrcrg1 = 12 xrcrg2 = 5
+cgbo = 2.56e-011 cgdl = 2.653e-10
+cgsl = 2.653e-10 ckappas = 0.03 ckappad = 0.03 acde = 1
+moin = 15 noff = 0.9 voffcv = 0.02

+kt1 = -0.11 kt1l = 0 kt2 = 0.022 ute = -1.5
+ua1 = 4.31e-009 ub1 = 7.61e-018 uc1 = -5.6e-011 prt = 0
+at = 33000

+fnoimod = 1 tnoimod = 0

+jss = 0.0001 jsws = 1e-011 jswgs = 1e-010 njs = 1

+ijthsfwd= 0.01	ijthsrev= 0.001	bvs = 10	xjbvs = 1
+jsd = 0.0001	jswd = 1e-011	jswgd = 1e-010	njd = 1
+ijthdfwd= 0.01	ijthdrev= 0.001	bvd = 10	xjbvd = 1
+pbs = 1	cjs = 0.0005	mjs = 0.5	pbsws = 1
+cjsws = 5e-010	mjsws = 0.33	pbswgs = 1	cjswgs = 3e-010
+mjswgs = 0.33	pbd = 1	cjd = 0.0005	mjd = 0.5
+pbswd = 1	cjswd = 5e-010	mjswd = 0.33	pbswgd = 1
+cjswgd = 5e-010	mjswgd = 0.33	tpb = 0.005	tcj = 0.001
+tpbsw = 0.005	tcjsw = 0.001	tpbswg = 0.005	tcjswg = 0.001
+xtis = 3	xtid = 3		
+dmcg = 0e-006	dmci = 0e-006	dmdg = 0e-006	dmcgt = 0e-007
+dwj = 0.0e-008	xgw = 0e-007	xgl = 0e-008	
+rshg = 0.4	gbmin = 1e-010	rbpb = 5	rbpd = 15
+rbps = 15	rbdb = 15	rbsb = 15	ngcon = 1