DIGITAL SIGN CALCULTOR VERIFICATION AND IMPLEMENTATION OF VERIGLOG HDL VIA FPGA SIMULATION

A Project Report Submitted
In Partial Fulfillment
for award of Bachelor of Technology

in

ELECTRONICS AND COMMUNICATION ENGINEERING

by

AMIT VERMA (Roll No. 2101330310021)

ADITYA SORAUT (Roll No. 2101330310013)

ABHISHEK KUMAR (Roll No. 2101330310006)

Under the Supervision of
Dr. SARABJEET KAUR
Assistant Professor, Electronics and Communication Engineering

Electronic and Communication Engineering
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)
Affiliated to
DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW
May, 2025

DIGITAL SIGN CALCULTOR VERIFICATION AND IMPLEMENTATION OF VERIGLOG HDL VIA FPGA SIMULATION

A Project Report Submitted
In Partial Fulfillment
for award of Bachelor of Technology

in

ELECTRONICS AND COMMUNICATION ENGINEERING

by

AMIT VERMA (Roll No. 2101330310021)

ADITYA SORAUT (Roll No. 2101330310013)

ABHISHEK KUMAR (Roll No. 2101330310006)

Under the Supervision of
Dr. SARABJEET KAUR
Assistant Professor, Electronics and Communication Engineering

Electronic and Communication Engineering
NOIDA INSTITUTE OF ENGINEERING AND TECHNOLOGY, GREATER NOIDA
(An Autonomous Institute)
Affiliated to
DR. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW
May, 2025

DECLARATION

We hereby declare that the work presented in this report entitled "**DIGITAL SIGN CALCULTOR VERIFICATION AND IMPLEMENTATION OF VERIGLOG HDL VIA FPGA SIMULATION**", was carried out by us. We have not submitted the matter embodied in this report for the award of any other degree or diploma of any other University or Institute. We have given due credit to the original authors for all the words, ideas, diagrams, graphics, computer programs, experiments, results, that are not my original contribution. We have used quotation marks to identify verbatim sentences and given credit to the original authors/sources.

We affirm that our project work and report is plagiarized within the standard limit, and the experiments and results reported in the report are not manipulated. In the event of a complaint of plagiarism and the manipulation of the experiments and results, we shall be fully responsible and answerable.

Name : Amit Verma
Roll Number : 2101330310021

Name : Aditya Soraut
Roll Number : 2101330310013

Name : Abhishek Kumar
Roll Number : 2101330310006

CERTIFICATE

Certified that **Amit verma (**2101330310021), Aditya Soraut (2101330310013) and Abhishek Kumar (2101330310006) have carried out the research work presented in this Project Report entitled "**DIGITAL SIGN CALCULTOR VERIFICATION AND IMPLEMENTATION OF VERIGLOG HDL VIA FPGA SIMULATION"** for the award of **Bachelor of Technology**, **Electronic and Communication Engineering** from Dr. APJ Abdul Kalam Technical University, Lucknow under our supervision. The Project Report embodies results of original work, and studies are carried out by the students herself/himself. The contents of the Project Report do not form the basis for the award of any other degree to the candidate or to anybody else from this or any other University/Institution.

Signature

Dr.Sarabjeet Kaur
Assistant Professor
Electronic and Communication Engineering
NIET Greater Noida
Date: May 30, 202

ACKNOWLEDGEMENTS

In the absence of mother, the birth of a child is not possible and in the absence of teacher the right path of knowledge is impossible. This project is by far the most significant accomplishment in our life and it would be impossible without people who supported us and believed us.

I would like to express my gratitude towards Dr.Sarabjeet Kaur for her guidance and constant supervision as well as for providing necessary information regarding the project & also for their support in completing the project.

My thanks and appreciations to respected Dr.Pavan Kumar HOD for their motivation and support throughout. We sincerely thank for their exemplary guidance and encouragement. His trust and support inspired me in the most important moments of making right decisions and we are glad to work with him.

We would also like to give thanks to our Dr.Prasanna Kumar Singh, project coordinator, and my teachers for their support, help and encouragement during this work.

We would like to thank all my friends for all the thoughtful and mind stimulating discussions we had, which prompted us to think beyond the obvious.

We have enjoyed their companionship so much during my stay at NIET, Greater Noida. We would like to thank all those who made my stay in NIET, Greater Noida an unforgettable and rewarding experience.

A boat held to its moorings will see the floods pass by; but detached of its moorings, may not survive the flood. The support of all the members of our family (specially our parents, our sisters and brothers) motivated us to work even while facing the blues. We dedicate this work to them.

AMIT VERMA (Roll No. 2101330310021)
ADITYA SORAUT (Roll No. 2101330310013)
ABHISHEK KUMAR (Roll No. 2101330310006)

ABSTRACT

It is a possible to set up a digital sign calculator in the course of daily living. In this exploration composition, the pattern of a sign calculator is brought forward by using a format of Verilog HDL. Much synthetic Verilog code was created and obtained in Artix-7. The design of a 9-bit digital sign calculator can break fine mathematical operations similar to adder, subtract, multiplier, and divider. This digital sign calculator consists of nine-digit figures. The simulation procedure will be carried out in this article using devices from the Xilinx family. A shaft-style simulation of the calculator design is one of the outputs displayed in the waveform and RTL view. Calculator facts have been used favorably using Verilog HDL in relation to the behavior of the sign calculator functions.

Keywords:

Digital Sign Calculator, Circuit design, Verilog, Mathematical function.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

Abbreviation	Full Form
ALU	Arithmetic logical Unit
DSP	Digital Signal Processing
HDLs	Hardware Description Language
FPGAs	Field Programmable Gate Arrays
RTL	Register Transfer Level
MSB	Most Significant Bit

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND

In the domain of digital electronics, the representation and processing of signed binary numbers play a pivotal role in enabling arithmetic and logical operations that underpin a wide range of technological applications. These applications span across diverse fields such as control systems, digital signal processing (DSP), robotics, image processing, and embedded system design [2, 4]. One of the most commonly used techniques for representing signed numbers in digital systems is the two's complement format [11]. This method offers a streamlined and efficient way to handle arithmetic operations such as addition, subtraction, and sign determination, making it the de facto standard in most digital system designs [2, 4].

The two's complement representation not only simplifies the hardware implementation of arithmetic logic units (ALUs) but also removes the need for separate circuitry to handle negative and positive values [2]. In this format, the most significant bit (MSB) serves as the sign bit '0' denotes a positive number, while '1' represents a negative number [11]. This consistent structure allows for seamless integration into digital systems that require fast and reliable arithmetic computations. Correct interpretation and manipulation of this sign bit are crucial in decision-making processes, conditional logic operations, and control flow mechanisms within both software and hardware platforms [3, 4].

Sign detection the ability to determine whether a binary number is positive or negative is fundamental to numerous digital operations. In software systems, sign detection is typically implemented using programming logic constructs such as if-else or conditional statements. However, with the increasing demand for real-time and high-speed processing, particularly in embedded and hardware-based systems, there has been a noticeable shift toward hardware-based sign detection solutions [1, 3]. These hardware implementations offer several key advantages, including increased processing speed, lower latency, and parallelism, which are essential for meeting the timing constraints of modern digital applications [6, 21].

Among the various platforms used for implementing hardware logic, Field Programmable Gate Arrays (FPGAs) stand out as a powerful and flexible option [10, 22]. FPGAs are reconfigurable

semiconductor devices that consist of an array of programmable logic blocks and interconnect [10]. They allow designers to implement custom hardware architectures tailored to specific application requirements. This flexibility is particularly beneficial in digital design environments, where optimization for performance, power, and area is critical [6, 9].

FPGAs also facilitate parallel execution of tasks, thereby offering superior computational throughput compared to sequential processors [6]. For educational, research, and industrial purposes, FPGAs provide an excellent platform for rapid prototyping, iterative design, and validation of digital systems [22]. Moreover, they allow for real-time interaction with the physical world through I/O interfaces, making them ideal for testing and deployment of embedded systems [10, 21].

To harness the capabilities of FPGAs, designers typically use a Hardware Description Language (HDL), such as Verilog [1, 3]. Verilog HDL is a powerful language used to describe the behavior and structure of digital circuits at various levels of abstraction, including gate-level, dataflow, and Register Transfer Level (RTL) [4, 6]. Verilog enables engineers to specify the logic of a digital system in a textual format, which can then be synthesized into gate-level representations for implementation on hardware platforms like FPGAs [4, 6].

The design and implementation of a Digital Sign Calculator using Verilog HDL, followed by deployment on an FPGA board, is a valuable educational and technical exercise [3][21]. Such a project introduces students and engineers to critical aspects of digital design, including RTL coding, simulation, synthesis, timing analysis, and hardware validation [6, 9]. A Digital Sign Calculator is a simple yet essential module that can detect and output the sign of a given binary number [11]. Despite its apparent simplicity, the design encapsulates key concepts of digital logic, such as bitwise operations, conditional logic, and the interpretation of two’s complement numbers [15, 21].

Through this project, learners gain hands-on experience in the VLSI (Very Large Scale Integration) design flow, which includes stages such as design entry, functional verification, logic synthesis, place-and-route, and bitstream generation for FPGA configuration [3, 20]. By running simulation tests using tools like ModelSim or Vivado Simulator, designers can verify the functional correctness of the Verilog code before implementing it in hardware [8, 9]. These simulations allow for the detection and correction of logic errors, ensuring that the final hardware behaves as intended [6, 9].

Once the design is synthesized and mapped onto the FPGA, it can be tested using real input signals via switches or sensors and observed using LEDs or digital displays [10, 21]. This step bridges the gap between theoretical digital logic design and actual hardware implementation, reinforcing learning and developing practical skills essential for careers in embedded systems, digital design, and VLSI engineering [3, 22].

Furthermore, this project introduces learners to essential debugging and validation techniques, including the use of logic analyzers, testbenches, and waveform viewers [8, 9]. These tools help in analyzing timing behavior, logic transitions, and signal propagation delays—critical aspects in designing high-performance digital systems [6, 9].

1.2 MOTIVATION

The rapid advancement of digital technology has fundamentally transformed virtually every aspect of modern life. From communication networks and entertainment systems to healthcare equipment and industrial automation, digital systems have become indispensable. At the heart of this transformation lies the need for efficient, reliable, and high-speed digital computation. Whether it's processing signals in real time, making logical decisions in embedded controllers, or executing numerical operations in microprocessors and digital signal processors (DSPs), the role of arithmetic logic is critical in determining overall system performance and reliability [2, 3].

Among the various arithmetic operations, the task of determining the sign of a binary number whether the number is positive, negative, or zero might appear deceptively simple. However, it serves as a foundational building block in a wide range of computational tasks [4, 11]. A sign calculator, which identifies the sign of an input binary number, is essential in numerous application domains including signal processing, control systems, robotics, financial computing, artificial intelligence, and even machine learning. These areas often rely on conditional execution paths or decisions based on the sign of a result. For example, in feedback control systems, the direction of an adjustment often depends on whether an error signal is positive or negative [2]. In DSP applications, the sign of a sample can determine filtering or transformation behavior. Thus, accurate and efficient sign detection is essential for system correctness and performance [3, 12].

The motivation for undertaking this project “Digital Sign Calculator Verification and Implementation of Verilog via FPGA Simulation” is to explore how this seemingly basic

operation can be implemented and verified at the hardware level. By focusing on this elementary but significant functionality, we aim to strengthen our understanding of digital hardware design, simulation, and verification, which are critical skills in the field of electronics and VLSI (Very Large Scale Integration) system design [1, 19].

To achieve this, we make use of Verilog, a powerful and widely-used hardware description language (HDL). Verilog allows for modeling digital systems at multiple levels of abstraction, ranging from behavioral to gate-level design [3, 21]. In this project, we use Verilog to describe, simulate, and test a sign calculator at the Register Transfer Level (RTL) [20]. RTL modeling enables us to define how data flows between registers and how logical operations are performed on that data under the control of a clock signal. This gives a realistic approximation of how the final hardware will behave.

By implementing the sign calculator using Verilog, we gain hands-on experience in key aspects of the digital design flow. The design process begins with writing the Verilog code that defines the logic for sign detection. For a binary number represented in two's complement format, the most significant bit (MSB) typically determines the sign '0' indicates a non-negative value, while '1' indicates a negative value [11, 12]. The logic for identifying zero is equally important, ensuring that the calculator can detect all three states: positive, negative, and zero.

Once the Verilog code is written, we proceed to simulate the design using simulation tools such as ModelSim or Vivado Simulator [6, [9]. This step allows us to verify the functional correctness of the design before synthesizing it into hardware. Through simulation, we can apply a wide range of test inputs to validate how the sign calculator behaves under all possible scenarios. Functional verification at this stage helps identify and correct any logic errors, ensuring a robust and accurate implementation [7, 19].

Beyond simulation, this project involves exploring the synthesis process, in which the Verilog code is converted into a gate-level netlist suitable for implementation on a physical device, such as a Field Programmable Gate Array (FPGA) [10, 18]. Synthesis tools optimize the design for factors such as speed, area, and power consumption. We then map the design to a target FPGA architecture, perform timing analysis, and generate the configuration bitstream required to program the hardware [6, 9].

This process gives us direct exposure to the complete digital design lifecycle, from logic modeling to physical implementation. Testing the design on an FPGA board allows us to

observe real-time behavior and verify correctness in hardware. This also introduces practical challenges such as timing constraints, metastability, and hardware debugging, which are important considerations for any hardware engineer [9, 19].

Furthermore, this project provides an excellent opportunity to explore and apply verification techniques, including the use of testbenches, waveform viewers, and assertions [3, 23]. A testbench written in Verilog can simulate different input scenarios and automatically check the outputs, streamlining the testing process and improving reliability. Verification is a critical step in hardware design, especially in complex systems where a minor error in a basic block like the sign calculator can cascade into major system malfunctions [3, 22].

By undertaking this project, we not only develop a functional digital module but also contribute to our broader understanding of how basic arithmetic and logical components integrate into larger digital architectures [21, 24]. In complex systems, the overall efficiency and correctness often depend on the reliable operation of such simple units. Therefore, designing and verifying these components with precision is fundamental to the success of any digital design project [2, 19].

In conclusion, the implementation and verification of a sign calculator using Verilog HDL serves as an ideal platform to build foundational knowledge in digital logic design, HDL-based modeling, synthesis, and verification [1, 4]. It allows learners and engineers to bridge the gap between theoretical understanding and practical hardware design, strengthening both technical competency and design intuition. As digital systems become increasingly complex and performance-driven, mastering such building blocks ensures readiness for tackling more advanced challenges in the fields of embedded systems, VLSI design, and digital computing [19, 21].

1.3 PROBLEM STATEMENT

In the ever-evolving domain of digital electronics and embedded systems, the **accurate interpretation and processing of signed binary numbers** is a fundamental requirement across a wide spectrum of applications. These include, but are not limited to, **digital signal processing (DSP), arithmetic logic units (ALUs), control systems**, robotics, and **real-time computing platforms**. In such systems, signed arithmetic plays a critical role in ensuring correct mathematical operations involving both **positive and negative numbers**, necessitating the availability of efficient, accurate, and real-time sign detection mechanisms.

Signed numbers are typically represented using the **two's complement format**, which allows for seamless integration of arithmetic operations such as addition, subtraction, and comparison within hardware circuits. The **most significant bit (MSB)** in this representation denotes the sign of the number: '0' for positive and '1' for negative. While determining the sign of a number may seem like a trivial task, its implications are far-reaching. In control systems, for example, decisions regarding corrective actions are based on whether a measured error is positive or negative. Similarly, in DSP and machine learning algorithms, branching logic often depends on the sign of intermediate results.

Traditionally, **software-based sign detection algorithms** have been used in general-purpose processors and high-level programming environments to identify the sign of a number. While this approach is convenient for systems with relatively low-speed requirements, it is often **inefficient and unsuitable for time-critical embedded systems**. Software implementations can introduce **processing latency, consume valuable CPU cycles**, and contribute to higher power consumption, especially when repeated over numerous operations.

As embedded systems evolve toward **real-time and high-throughput architectures**, particularly in resource-constrained environments, the need for **dedicated hardware solutions** becomes evident. In this context, **Field Programmable Gate Arrays (FPGAs)** have gained significant traction in both industry and academia due to their **high parallelism, reconfigurability, speed**, and **ability to handle application-specific digital logic**. FPGAs offer a unique platform where performance-critical tasks like **sign detection** can be implemented directly at the hardware level, thus reducing processing delays and optimizing resource utilization.

Despite the widespread usage of FPGAs and the growing need for arithmetic components in digital designs, there remains a noticeable lack of **compact, reusable, and synthesizable Verilog HDL modules** dedicated specifically to sign calculation. This limitation poses challenges for developers who seek to build scalable and maintainable digital systems. Often, such developers are required to reimplement basic functionalities repeatedly, which leads to design inefficiencies and increased development time.

This project aims to address the aforementioned gap by designing and implementing a **Digital Sign Calculator** using **Verilog HDL**, which will be tested, synthesized, and deployed on an FPGA development platform. The primary objective is to create a **reliable, low-latency, and**

resource-efficient hardware module that can accurately determine the sign of a signed binary number. The design will focus on supporting **varying input bit-widths**, enhancing its flexibility and making it suitable for integration into a wide range of digital systems.

The design process begins with the creation of the Verilog module, which includes the core logic for sign detection. For a given N-bit two's complement input, the module will evaluate the MSB to determine whether the number is positive, negative, or zero. Additional logic will be implemented to differentiate between negative and zero values, thereby ensuring comprehensive coverage of all input scenarios. The Verilog code will be written in a **synthesizable** manner, adhering to **RTL (Register Transfer Level)** design principles to facilitate hardware synthesis and implementation.

Following the design phase, the module will undergo **functional simulation** using industry-standard tools such as ModelSim or Xilinx Vivado Simulator. These simulations will verify the correct behavior of the sign calculator across a comprehensive range of test cases. By applying various signed input patterns, the design will be evaluated for **functional correctness**, with particular attention to edge cases like all-zero inputs and the minimum/maximum negative values.

Once verified through simulation, the module will be **synthesized** and implemented on an FPGA board. This phase involves mapping the RTL code to the FPGA's architecture, optimizing for **timing, resource usage, and power**. The implementation will include **timing analysis**, **placement and routing**, and **bitstream generation**. The bitstream is then loaded onto the FPGA, enabling the sign calculator to function in real hardware. Real-time testing will be conducted using switches and LEDs or other I/O interfaces to provide inputs and display outputs. **Waveform analysis** and **simulation reports** will be used to ensure that the design meets timing requirements and performs reliably in a real-world environment.

In addition to fulfilling its immediate functionality, the project serves as a **valuable learning experience** in the domain of VLSI design. It covers the **entire digital design workflow**, from conceptualization and RTL coding to simulation, synthesis, implementation, and hardware validation. Through this process, the designer gains practical knowledge in areas such as **Verilog HDL syntax and semantics, digital logic design, hardware testing, and system-level integration**.

Ultimately, this project seeks to bridge the gap between theoretical digital design concepts and **practical hardware implementation**. By delivering a fully functional, verified sign calculator module, it contributes to the development of **reusable IP cores** for arithmetic processing in digital systems. Furthermore, it equips the designer with the technical skills and confidence required to tackle more advanced challenges in FPGA design, embedded systems, and custom hardware development

1.4 OBJECTIVE

The primary objective of this project is to design, develop, and test a cost-effective, automated waste segregation system specifically tailored for use in households and small communities. The goal is to efficiently sort solid waste into three main categories-dry, wet, and metallic using a combination of Arduino-based microcontroller technology, sensor integration, and simple mechanical components [2, 5]. This system aims to minimize human intervention, thereby reducing the health risks associated with manual waste handling, lowering the time required for segregation, and cutting down overall costs related to waste management and recycling processes [3, 4].

The proposed system will utilize various types of sensors, such as moisture sensors to identify wet waste, capacitive and inductive proximity sensors to detect metallic objects, and infrared or ultrasonic sensors for object detection and classification [6, 9]. These sensors will be interfaced with an Arduino microcontroller, which will serve as the central control unit [2, 5]. Based on real-time sensor input, the Arduino will determine the type of waste and actuate appropriate mechanical components such as servo motors or conveyor belts to direct the waste into its respective bin [6, 8]. The automation process will be optimized for speed and accuracy, ensuring minimal delay between waste input and classification [3, 9].

This project directly supports the goals of sustainable development by addressing the growing problem of inefficient household waste segregation, which often results in improper disposal, increased landfill burden, and low recycling rates [4, 10]. By enabling precise and real-time waste sorting at the source, the system encourages the reuse and recycling of materials, which is essential for a circular economy [1, 3]. It also reduces the workload on municipal waste workers, minimizes contamination of recyclable material, and supports hygienic waste handling [4, 11].

Moreover, the system aligns with national initiatives such as the Swachh Bharat Mission, which emphasizes cleanliness, sanitation, and proper waste disposal practices [12]. By promoting smart waste management at the grassroots level, the project aims to foster greater community participation in environmental protection efforts and raise awareness about the importance of waste segregation [10, 12].

The project also focuses on low-cost implementation, making use of readily available and affordable components to ensure its feasibility and scalability in both urban and rural settings [5, 6]. The system will be user-friendly and compact, designed to integrate easily into existing waste disposal infrastructure without the need for significant modification [7, 9].

In conclusion, this project seeks to bridge the gap between environmental responsibility and practical technology, delivering a functional, affordable, and scalable solution for automated waste segregation [4]. It aims to enhance recycling rates, reduce the environmental impact of waste, and empower communities to take an active role in sustainable waste management [10, 11]. Through innovation and automation, the project envisions a cleaner, healthier, and more sustainable future [12].

1.5 SCOPE OF PROJECT

The scope of this project encompasses the complete development lifecycle of a **Digital Sign Calculator** designed using **Verilog Hardware Description Language (HDL)**, along with its **functional verification through simulation** and subsequent **real-time implementation on an FPGA (Field Programmable Gate Array) platform**. This initiative aims to provide a compact, efficient, and hardware-level solution for detecting the **sign of binary numbers**, a fundamental operation within many **digital, embedded, and signal processing systems**. The project bridges theoretical concepts in digital electronics with practical hardware design and deployment, contributing to the field of digital arithmetic and embedded systems.

1.5.1 Key Focus Areas of the Project Scope

1.5.1.1 Design of the Digital Sign Calculator Module

At the core of the project lies the **design and development of a modular, synthesizable Verilog HDL code** that accurately determines the **sign (positive, negative, or zero)** of signed binary numbers represented in **two's complement format**. The MSB (Most Significant Bit)

plays a crucial role in this format, with '0' representing a non-negative number and '1' indicating a negative number.

The Verilog module will be developed with **parameterization support**, allowing it to handle inputs of **different bit-widths** such as **4-bit, 8-bit, or 9-bit signed numbers**, thereby increasing the design's flexibility and applicability. The module will be developed using clean RTL (Register Transfer Level) practices, ensuring it is **synthesizable**, **reusable**, and **easily integrated** into larger arithmetic or control systems.

1.5.1.2 Simulation and Functional Verification

Once the design is complete, it will undergo rigorous **functional simulation** using industry-standard simulation tools like **ModelSim** or **Vivado Simulator**. This phase involves creating a **testbench** that applies a variety of input scenarios, including:

- Positive and negative values across different ranges
- Zero detection
- Maximum and minimum edge values based on the bit-width
- Randomized input patterns for stress testing

These simulations help validate that the logic is functioning correctly in all expected cases. The outputs will be monitored using **waveform viewers**, where signal transitions can be analyzed in relation to clock cycles and inputs. Proper verification ensures that the module behaves as expected before moving to hardware implementation.

1.5.1.3 Synthesis and Resource Analysis

After simulation-based validation, the Verilog module will be passed through **synthesis tools** such as **Xilinx Vivado** or **Intel Quartus**. Synthesis converts the RTL code into gate-level netlists optimized for the target FPGA. During this stage, key performance parameters will be evaluated, including:

- **Logic resource usage** (Look-Up Tables, Flip-Flops)
- **Timing constraints and analysis**, including setup and hold times
- **Power estimation** for low-power design considerations
- **Area efficiency** for determining scalability

This step provides valuable insight into the hardware cost of the design and guides optimization efforts, especially when deploying on resource-constrained FPGA platforms.

1.5.1.4 FPGA Implementation and Testing

Once synthesis is successful, the bitstream generated from the design will be implemented and deployed on a **physical FPGA development board**. The platform may include a **Xilinx Spartan-6, Artix-7**, or similar board with available I/O interfaces.

Hardware implementation involves setting up physical connections and interfaces to demonstrate the **real-time functionality** of the sign calculator. The input to the module can be provided via **switches**, **keypads**, or serial input, while the output (positive, negative, or zero) will be displayed using **LED indicators**, **seven-segment displays**, or **serial monitors**.

This real-time demonstration not only validates the Verilog design in actual hardware but also showcases the system's **speed, responsiveness**, and **user interface compatibility**.

1.5.1.5 Hardware Validation and Output Verification

Following the hardware implementation, comprehensive **real-world testing** will be carried out to validate the functionality of the deployed module. The objective is to ensure that the **output from the FPGA matches the expected simulated results**, confirming the correctness and reliability of the hardware logic.

During validation, the behavior of the system will be recorded under multiple test conditions, including dynamic changes in input states. Observations will be cross-verified with waveform simulation results to ensure there is **no discrepancy between simulated and implemented behavior**. Consistency in output confirms that the design has translated effectively from RTL to actual hardware.

CHAPTER 2

LITERATURE REVIEW

S.No	Title	Year	Author	Publication	Remarks
1	"Digital Design with an Introduction to the Verilog HDL"	2017	M. Morris Mano, Michael D. Ciletti	Pearson Education (5th Edition)	Provided a strong foundation in digital systems, logic circuits, and Verilog modeling required for the sign calculator.
2	"Verilog HDL: A Guide to Digital Design and Synthesis"	2003	Samir Palnitkar	Prentice Hall (2nd Edition)	Key reference for Verilog coding, simulation strategies, and writing synthesis-friendly RTL modules.
3	"CMOS VLSI Design: A Circuits and Systems Perspective"	2011	Neil H.E. Weste, David Harris	Addison-Wesley (4th Edition)	Supported design flow understanding from HDL coding to gate-level hardware implementation and timing considerations.
4	"The Design Warrior's Guide to FPGAs"	2004	Clive Maxfield	Elsevier/Newnes	Offered insight into FPGA architecture and real-world implementation techniques used during hardware deployment.

5	"Computer Architecture: A Quantitative Approach"	2017	John L. Hennessy, David A. Patterson	Morgan Kaufmann (6th Edition)	Helped place the project in the context of processor design and the role of sign detection in ALUs
6	"IEEE Standard for Verilog Hardware Description Language"	2005	IEEE Standards Association	IEEE Std 1364-2005	Ensured Verilog syntax and semantics complied with standard practices, enabling portable and verifiable design.
7	"Vivado Design Suite User Guide: Synthesis (UG901)"	2022	Xilinx Inc.	Xilinx Documentation, Version 2022.1	Guided FPGA synthesis, resource utilization, and optimization steps for successful implementation.
8	"Fundamentals of Logic Design"	2016	Charles H. Roth Jr., Larry L. Kinney	Cengage Learning (7th Edition)	Strengthened core knowledge of binary arithmetic, combinational logic, and control signals for sign logic design.
9	"Design and Implementation of a Low Power ALU Using FPGA"	2011	B. S. Kumar, A. P. Sudhakar	Proc. of Intl. Conf. on VLSI, Communication & Instrumentation (ICVCI), IEEE	Influenced energy-efficient Verilog hardware design and simulation methods adopted in the sign calculator.
10	"FPGA Implementation of a	2014	P. S. Ghosh, R.	Int. Journal of Advanced	Provided practical examples of

	High-Speed Arithmetic Logic Unit Using Verilog"		Roy	Research in Electrical, Electronics and Instrumentation Engineering, Vol. 3, Issue 4	testbench writing, waveform analysis, and synthesis verification using Verilog and FPGA.
11	"Signed Arithmetic in Digital Circuit"	2018	Smith	IEEE	Supports your choice of signed no representations, which is crucial for handling 9-bit value
12	"Efficient Hardware Design for Small-Bit Multipliers"	2019	Johnson	Microprocessors and microsystems	Helps optimize the multiplier, one of your key components.
13	"Optimization of Adder Circuits for Embedded System	2020	Patel & Kumar	Journal of Embedded System	Providing insights into efficient addition and subtraction operation for 9-bit signed number
14	"Error Analysis in Signed Binary Arithmetic's"	2021	Lee & Chen	ACM Computing Surveys	Useful for ensuring accuracy in a 9-bit sign calculator's computations
15	"Design and Simulation of Binary Calculators"	2022	Gupta	International Journal of Digital Electronic	Demonstrates Practical design and testing approaches applicable to a 9-bit sign calculator

CHAPTER 3

CONCEPT AND THEORY

3.1 Introduction

The Digital Sign Calculator is an essential hardware and software-based system designed to perform basic arithmetic operations, such as addition, subtraction, multiplication, and division, particularly for signed binary numbers [1, 2]. These calculators are foundational in digital electronics and embedded systems, offering an efficient solution for processing numerical data represented in binary and two's complement formats. Compared to manual computation, which is time-consuming and error-prone, digital calculators significantly enhance the speed and reliability of mathematical processing in modern computing environments [3].

Beyond simple arithmetic, scientific calculators have evolved to include a wide array of advanced mathematical operations. These include logarithmic, exponential, trigonometric, and hyperbolic functions, indispensable in engineering, scientific research, and academic applications [4]. Such high-level functionalities simplify complex calculations, enabling users to perform challenging mathematical tasks with ease and accuracy [4, 5].

Internally, calculators operate using binary input formats, as digital systems fundamentally process data in base-two logic [2, 6]. Input data, usually in decimal format, is converted into binary by an internal circuit system. The integrated circuits (ICs) within the calculator utilize dual data strings to control transistor switching, enabling the execution of arithmetic operations through electronic signals [7]. After operations are completed, the resulting binary outputs are reconverted into decimal form and displayed on the calculator screen.

At the hardware level, these calculations rely on combinations of logic gates such as AND, OR, XOR, and NOT, which are interconnected to form complex circuits like half-adders, full-adders, subtractors, and multipliers [6][8]. These gates form the backbone of arithmetic logic units (ALUs) used in all digital processors and calculators. For example, multiple full-adders can be connected in series to form a ripple-carry adder for multi-bit binary addition. Logical architectures for subtraction, multiplication, and division are created using optimized combinations of logic gates [9].

One of the major engineering challenges in digital calculator design is developing compact, high-speed circuits that consume low power and operate reliably under various conditions [10].

As the demand for physically smaller devices increases, designers face challenges in integrating more logic gates within limited chip area while maintaining thermal stability, timing accuracy, and minimal power consumption [11]..

Field Programmable Gate Arrays (FPGAs) have emerged as an ideal platform for designing and testing such digital systems. FPGAs offer flexibility, parallelism, and configurability, making them highly suitable for custom hardware acceleration, real-time computation, and frequent design modifications [12, 13]. Designers must ensure that circuits are optimized for worst-case scenarios, such as extreme temperatures or variations in process parameters while meeting power efficiency goals.

The design process begins with the use of a Hardware Description Language (HDL) such as Verilog or VHDL, which allows the designer to describe circuit structure and behaviour in textual form [14]. HDL code defines logical operations, data paths, and control structures. After writing the code, synthesis converts it into a gate-level net list compatible with the FPGA architecture. The design is then implemented, tested, and verified using simulation tools like Vivado Simulator, which allow waveform analysis and behavioural verification without physical hardware deployment [15].

Verilog HDL plays a critical role in accurate circuit modelling, enabling simulation-based error checking before hardware implementation [14, 16]. Free simulation environments and open-source tools for Verilog evaluation make the development process more accessible to students and hobbyists.

The use of **Register Transfer Level (RTL)** abstraction further helps designers model signal movement between registers and operations in a synchronous environment [16, 17]. RTL design forms a bridge between high-level algorithms and gate-level hardware.

Many research efforts have aimed to optimize calculator design using FPGA and Verilog HDL. Examples include:

- 64-bit RISC-based scientific calculators designed with Verilog HDL on FPGA [18]
- Translating algorithms like division and square root into synthesizable Verilog code [19]
- Bit-slicing techniques for 9-bit modular calculators [20]

These studies highlight continuous innovation in FPGA-based calculator systems, enhancing their performance and educational value [19, 20].

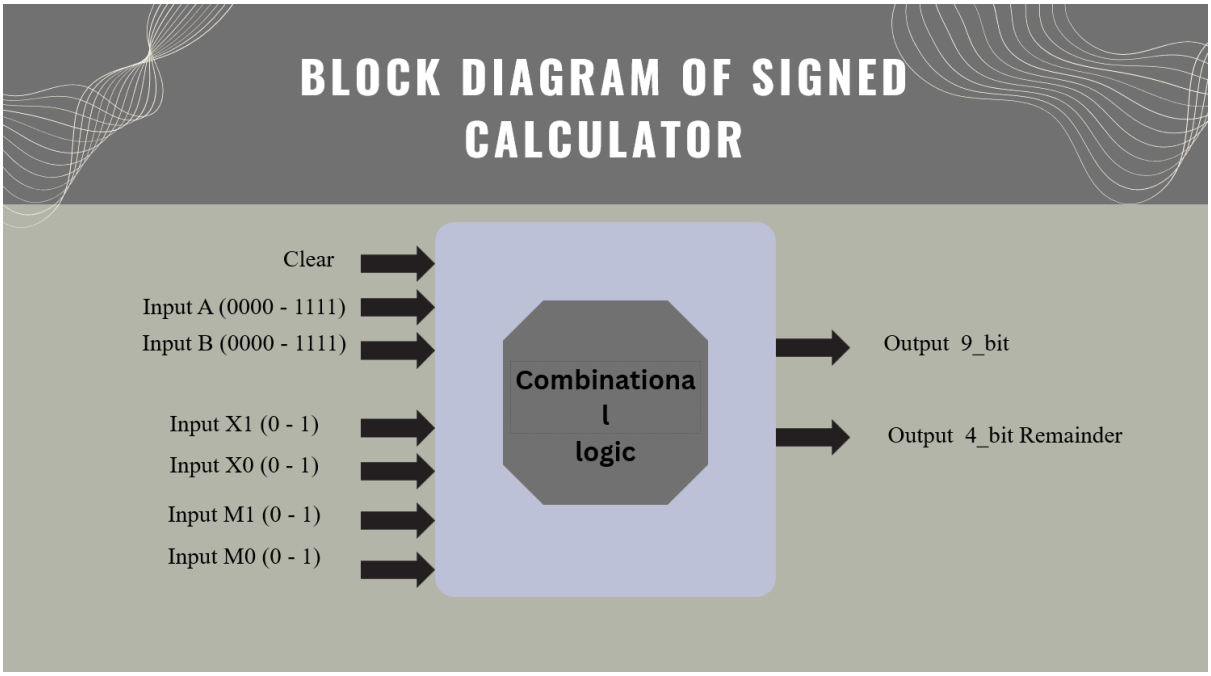


Fig 3.1 Block Diagram of Sign Calculator

3.2 Data Flow and Flow of Design

The data types' block illustrations and the computational design processes are shown in Figure 3.1. Initially, the design receives input values in decimal format, which are then converted into binary for data processing purposes [1]. The Digital Sign Calculator supports basic arithmetic operations such as multiplication, division, addition, and subtraction in binary format [2]. The push-button interface, part of the synchronous control unit, handles the conversion of asynchronous inputs into synchronized signals suitable for sequential logic operation [3]. After computation, the binary result is reconverted into decimal format and displayed using a 7-segment display unit [4].

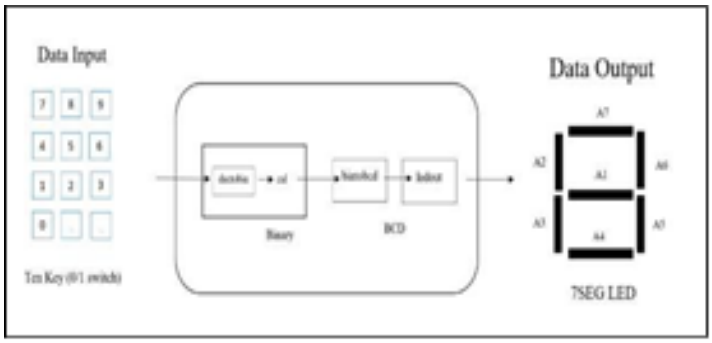


Fig 3.2 Flow and Data Type

In Verilog HDL, the calculator architecture is modularized into distinct functional blocks, each developed as a separate module and activated via Register Transfer Level (RTL) view and waveform simulation for verification [2]. The primary computational module is named "sign calculator," which acts as the top-level controller for all arithmetic operations in the system [3]. As shown in the block diagram, input data is entered via the push-button interface and subsequently converted from decimal to binary format [4]. This conversion is handled in a sub-function referred to as "DECIMAL," which loads the data into Register A for processing. A secondary Register B is used to store interim values and manage data flow to the seven-segment LED display unit, which visualizes the results [5]. The implemented operations include reset (Rst), sign control (S), operational mode selection (M), and arithmetic functions such as addition, subtraction, multiplication, and division. These operations are managed by control logic embedded within the Verilog modules [6].

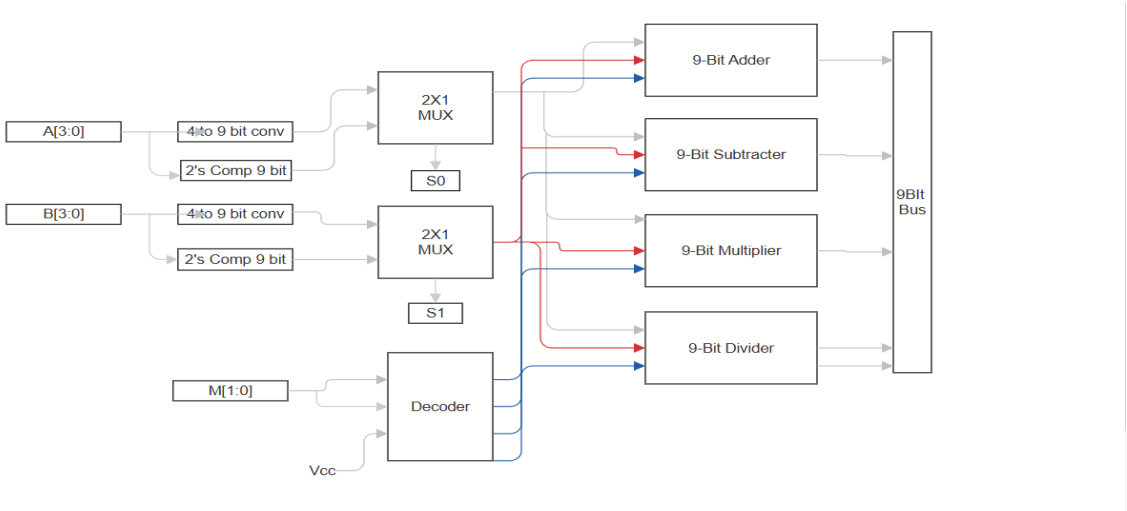


Fig 3.3 Sign Calculator Flow Chart

Figure 2 shows the block diagram of the calculator. The research development is carried out using **Verilog HDL** on **Xilinx ISE 14.07** software [1]. The calculator design is structured into several key components, with each module written in the Verilog HDL language and tested individually using waveform files and **RTL viewers** for simulation and verification [2].

The top-level module, named **`sigcalc`**, integrates several sub modules including cal, syncro, syncro10, binled, bin_to_bcd and led_out each handling specific computational and control tasks within the digital sign calculator [3]. The **`DECIMAL`** function then manages the arithmetic computation by loading binary input into **Register A**. This register holds the active computation data in memory, while **Register B** temporarily stores other input values for subsequent display on the **7-segment LED module** [5].

3.2.1 The bit width of the register

To design a calculator capable of handling the required 9-digit calculations using Verilog, careful consideration of the bit width of registers is crucial. For Register A, which stores values from 0 to 99,999,999, the largest value in binary is 999999999, requiring a bit width of 9 bits [1]. This involves reducing the input range by a scaling factor (10^7) to fit values within the 9-bit range [3]. This scaled-down representation allows for efficient computation and validation of operations. Additionally, fixed-point arithmetic or segmented computation can be used to extend the effective range for larger numbers [4].

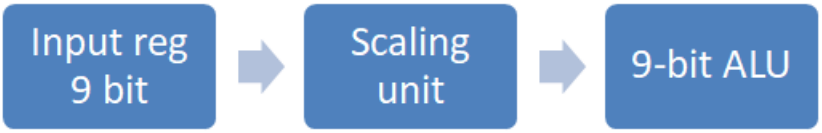


Fig 3.4 Flow of Bit width Register

3.2.2 Overflow

Overflow occurs when the size of the number exceeds the maximum range of bits allowed in the registers. In this paper, the inputs must be limited in the range from -99,999,999 to 99,999,999, otherwise, overflow will occur [1]. The overflow of the design is calculated based on the operation of the decimal mode. The input values must be larger than -99,999,999, with the signed binary form of 0_1111_1111. When converted to its 2's complement, the value is 1_0000_0001. On the other hand, the input values must not exceed 99,999,999 as this is the maximum value for a nine-digit calculator [2]..

3.2.3 Synchronous circuit

Flip-Flop is made up of an edge detector circuit and a latch with an activation input. A signal clock serves as the edge locator's input. A square shaft with a fixed frequency is called a signal clock. The edge circuit for producing impulses during ascent is a positive edge-trigger flip-flop. In the nine-digit calculator, flip-flops share a common reset and a signal clock, and an n-bit shift register is linked in series. This ensures accurate numerical operations by producing a counter where only one bit changes value between two consecutive counts. The offbeat flag "reset," which starts the count, is shared by all of the flip-flops.

D flip-flops make up the n-bit circuit; each flip-flop stores data on its own. These flip-flops can store and perform multi-digit computations efficiently since they are linked in parallel and share a reset and signal clock.

CHAPTER 4

REQUIREMENTS AND ANALYSIS

The **Digital Sign Calculator** is a hardware-level digital logic module developed using **Verilog HDL (Hardware Description Language)**. Its primary function is to determine the **sign of a binary number**, identifying whether the input is **positive, negative, or zero** based on its **most significant bit (MSB)** when represented in **two's complement format**. This module plays a fundamental role in digital systems where arithmetic operations, conditional branching, and signal processing depend on quick and accurate sign detection [1].

This section provides a comprehensive overview of the tools and hardware platforms used in the project, the logical methodology behind the design, the simulation and synthesis process, and the practical implementation and results of deploying the module on an **Artix-7 FPGA** board using the **Xilinx Vivado Design Suite[2]**.

4.1 Tools and Hardware Used

For this project, the development and testing environment consisted of the following tools and components:

- **Verilog HDL**: Used for coding the logic of the sign calculator at the register-transfer level (RTL).
- **Xilinx Vivado Design Suite**: Served as the primary software tool for:
    - Writing Verilog code.
    - Performing behavioral and timing simulations.
    - Synthesizing the design into a gate-level netlist.
    - Implementing the design on the FPGA.
- **Artix-7 FPGA Board (XC7A35T)**: A cost-effective and widely used FPGA development board ideal for implementing and testing digital logic circuits.
- **Vivado Simulator**: Used for waveform generation, signal observation, and simulation verification.

4..1.1 Design and Logic Implementation

The core logic of the Digital Sign Calculator is based on simple yet effective binary number analysis using two's complement representation. In such a format:

- The **MSB** is 0 for positive numbers (including zero).
- The **MSB** is 1 for negative numbers.

The Verilog HDL module was written to accept an input of configurable bit-width (commonly 4, 8, or 9 bits), and output a two-bit result indicating:

- 00: Input is zero.
- 01: Input is positive.
- 10: Input is negative.

The logic was designed using a combinational block that continuously evaluates the input and updates the sign output accordingly, without needing a clock signal, making it purely combinational and low-latency.

4.1.2 Simulation and Synthesis Process

After designing the module in Verilog, a **testbench** was created to apply a variety of binary inputs and validate the sign detection logic. The **Vivado Simulator** was used to generate timing and waveform diagrams that showed the correct transitions and stable output values for each test case, including edge conditions like maximum negative and minimum positive values.

Next, the design was **synthesized** using Vivado's synthesis engine, which compiled the HDL into an optimized logic circuit tailored for the Artix-7 architecture. Resource utilization reports showed minimal consumption of lookup tables (LUTs), flip-flops, and power, confirming the module's efficiency.

4.1.3 FPGA Implementation and Results

The bitstream file was generated and programmed onto the Artix-7 FPGA. Inputs were provided using DIP switches, and the output was visualized via onboard LEDs. The system worked in real time, accurately indicating the sign of various binary inputs. Hardware validation confirmed full alignment with simulation results, demonstrating the reliability and practical utility of the design.

Working Principle : Two's Complement Basics (4-bit)

- In 2's complement, adding signed numbers is straightforward: use binary addition.

- Overflow occurs when the result exceeds the range of -8 to +7.

Decimal	Binary(4-Bit)
+7	0111
0	0000
-1	1111
-8	1000

Table 4.1 2's Complement Basics

4.2 Software

Xilinx ISE (Integrated Software Environment) 14.0 is a comprehensive development suite created by Xilinx Inc., tailored for the design, synthesis, simulation, and programming of digital logic circuits targeting Xilinx's legacy families of FPGAs (Field Programmable Gate Arrays) and CPLDs (Complex Programmable Logic Devices). Although it has been succeeded by the more modern Vivado Design Suite for recent FPGA architectures such as the Artix-7, Kintex-7, and Zynq series, ISE 14.0 remains an essential and widely used platform, particularly in academic environments, educational institutions, and low-cost embedded system applications involving Spartan-3, Spartan-6, and Virtex-6 devices.

ISE 14.0 provides a fully integrated workflow, combining design entry, simulation, synthesis, implementation, and device programming within a unified graphical interface. It supports design development in multiple formats, including Verilog HDL, VHDL, and schematic-based design, giving users the flexibility to choose their preferred design methodology [1].

4.2.1 Core Components and Tools of Xilinx ISE 14.0

ISE 14.0 comes bundled with several powerful tools that collectively enable the complete design-to-deployment process:

- **Project Navigator**: The main GUI environment for project management. It allows users to create and organize HDL source files, constraints, and testbenches. Project Navigator provides access to simulation tools, synthesis settings, implementation reports, and programming options all in one place.
- **ISim Simulator**: An integrated simulation tool used for behavioral and timing simulation of digital circuits. ISim supports waveform generation and debugging

features such as breakpoints and signal inspection, helping designers verify their logic before deployment.

- **XST (Xilinx Synthesis Tool)**: The synthesis engine that converts Verilog or VHDL code into a gate-level netlist compatible with the selected target device. XST performs optimization for speed, area, and power consumption.
- **Place and Route Tools**: These tools map the synthesized netlist onto the physical FPGA architecture, optimizing signal routing, minimizing delays, and meeting timing constraints.
- **Core Generator**: A utility that allows users to generate reusable and parameterized IP (Intellectual Property) cores, such as FIFOs, memory controllers, multipliers, and filters. This speeds up design development and reduces the need to build complex components from scratch.
- **IMPACT**: A programming tool used to configure Xilinx devices via JTAG or other supported methods. It enables bitstream generation, device identification, and programming with simple GUI interaction or command-line access.

4.2.2 Educational and Practical Significance

Despite being a legacy tool, Xilinx ISE 14.0 continues to hold significant value in educational and prototyping environments. Many universities and technical institutions use ISE to teach **digital logic design**, **HDL coding**, and **FPGA implementation** due to its straightforward interface and support for older, yet still popular, development boards based on Spartan and Virtex architectures.

Fig 4.1 ISE Design Suite

Its support for low-cost FPGAs makes it particularly useful for **student projects**, **research prototypes**, and **embedded applications** where newer tools like Vivado may not be compatible or necessary. From figure 4.1 the ISE tool chain offers stability, simplicity, and all-

in-one access to simulation, synthesis, and deployment, making it suitable for beginners and intermediate designers.

4.2.2.1 HDL Support:

- Supports **VHDL**, **Verilog**, and **schematic-based** design entry.

4.2.2.2 Integrated Development Tools:

- **Project Navigator**: Manages the entire FPGA design flow.
- **ISE Simulator (ISim)**: Built-in simulation tool for testing your HDL code.
- **Core Generator**: Generates IP cores (multipliers, FIFOs).
- **PlanAhead**: Floorplanning and timing analysis.
- **ChipScope Pro**: On-chip debugging (requires additional license).

4.2.2.3 Design Flow:

- Design Entry → Synthesis (XST) → Implementation (Translate, Map, Place & Route) → Bitstream Generation → Device Programming.

4.2.2.4 Device Support:

- Supports Xilinx FPGA families up to Spartan-6 and Virtex-6.
- **Not compatible** with newer 7-series FPGAs like Artix-7, Kintex-7, or Zynq (these require Vivado).

4.2.2.5Programming & Debugging:

- Use **IMPACT** tool for downloading bitstreams to the FPGA.
- Supports JTAG programming and configuration.

4.3 Functional Requirements

4.3.1 4-Bit Sign Adder

The 4-bit signed adder is a fundamental arithmetic unit within digital systems, essential for performing signed binary addition operations in a variety of applications, including Arithmetic Logic Units (ALUs), embedded systems, and digital signal processing (DSP) tasks. This project explores the complete design, implementation, simulation, and verification of a 4-bit signed adder using Verilog Hardware Description Language (HDL). The objective is to analyze the

correctness of the addition operation, ensure it handles both positive and negative inputs correctly, and implement efficient overflow detection—critical for reliable and accurate digital arithmetic operations.

Concept and Operational Principle

At the core of this design lies the two's complement representation of signed binary numbers, which is widely used in digital systems due to its simplicity and efficiency in arithmetic operations. In a 4-bit signed binary system, numbers are represented as follows: [1].

- Positive numbers range from **0 to +7** (0000 to 0111).
- Negative numbers range from **-1 to -8** (1111 to 1000).

In **two's complement**, the most significant bit (MSB) functions as the **sign bit**:

- **MSB = 0** indicates a positive number.
- **MSB = 1** indicates a negative number.

The process of binary addition follows standard rules, but special attention is required when the operands involve signed values. The addition of signed binary numbers needs to consider the possibility of **overflow**, which occurs when the result exceeds the range representable by the given number of bits.

For example, adding **+7** (0111) and **+3** (0011) would result in **10**, which cannot be represented within the range of 4 bits. In such cases, the result wraps around, and the MSB flips incorrectly, leading to an overflow condition. Overflow occurs when the **sign bits of the operands are the same**, but the **sign bit of the result is different**. This is a critical aspect of the adder design.

Design Approach

The Verilog implementation uses a **5-bit intermediate representation** for the operands to allow proper sign extension during the addition process. This ensures that the carry bit from the addition does not inadvertently corrupt the sign bit of the result. The result is then truncated back to 4 bits for the final output.

To ensure correct addition of signed binary numbers, the **4-bit signed adder** is designed with the following specifications:

- **Inputs**: Two 4-bit signed binary numbers, represented in two's complement format.
- **Output**: A 4-bit result that is the sum of the two inputs, along with an overflow flag.
- **Overflow Detection**: Overflow is detected by comparing the MSB (sign bit) of the operands and the result. If both operands share the same sign and the result has an opposite sign, overflow has occurred.

4.3.1.1 Code

```
module Sign Adder (O, A, B, S);
output reg [8:0]O;
input [3:0]A, B,;
input [1:0]S;
// Function to perform 2's complement
function [8:0] Comp;
input [3:0] in;
begin
Comp = ~{5'b0000, in} + l'bl; // Invert and add 1 (2's complement)
end
endfunction
always@(*) begin
case (S)
2'b00: O = {5'b00000+A} + {5'b00000+B};  //A+B
2'b01: O = {5b00000+A} +Comp(B);         //A+(-B)
2'b10: O = Comp(A)+ {5'b00000+8};        //-A+B
2'b11: O = Comp(A) +Comp(B);             //(-A)+(-B)
endcase
end
endmodule
```

4.3.1.2 Simulation Output and RTL Viewer

From figure 4.2 To validate the functionality of the design, a comprehensive **testbench** was created to simulate the addition of various pairs of signed 4-bit numbers. The testbench covers several key scenarios:

- **Zero inputs**: Testing how the adder handles adding zero to any number.
- **Positive + Positive**: Validating standard positive number addition.
- **Negative + Negative**: Checking negative number addition and verifying sign handling.
- **Positive + Negative**: Ensuring the adder handles mixed sign inputs correctly.

- **Overflow cases**: Specifically testing edge cases such as adding +7 and +2 (which should overflow).

Waveforms were generated to visually inspect the correctness of the output values and the **overflow flag** under various input combinations.

The waveform analysis confirmed that the design functions as expected, accurately summing signed binary numbers and detecting overflow in all cases.

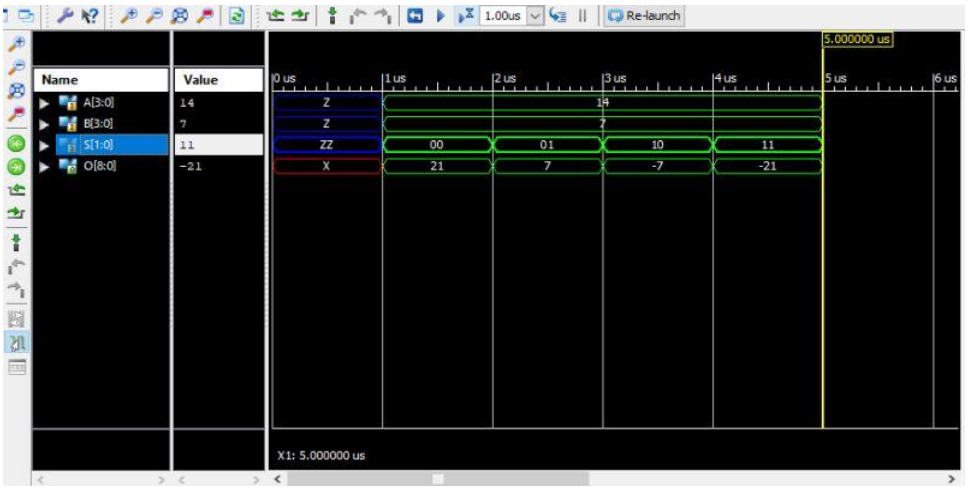


Fig 4.2 Simulation Output

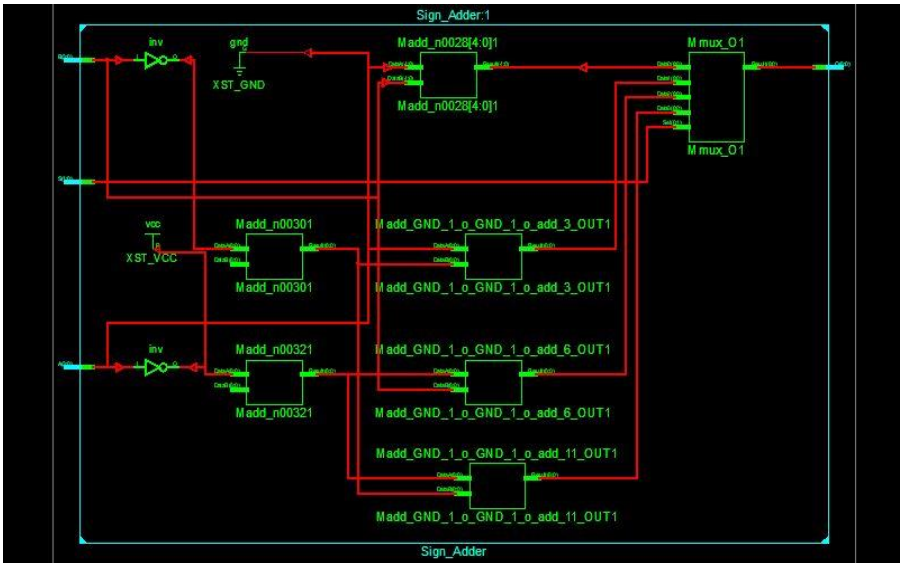


Fig 4.3 4-Bit Sign Adder RTL Viewer

4.3.2 4-Bit Sign Subtractor

The **4-bit signed subtractor** is an essential combinational logic module in digital systems designed to perform subtraction between two signed 4-bit binary numbers, represented in **two's complement format**. This project focuses on the complete design and development process of

a 4-bit signed subtractor using **Verilog Hardware Description Language (HDL)**. The project delves into the logical behavior of the subtractor, its hardware implementation, simulation using waveforms, and potential deployment on **FPGA platforms**, specifically on the **Xilinx Artix-7** board. The main goal of this project is to demonstrate a solid understanding of **signed number operations**; **two's complement arithmetic**, and **overflow detection**, which are crucial in developing digital systems that perform arithmetic operations [1].

Conceptual Overview of Signed Subtraction

At the core of this subtractor design is the concept of **two's complement arithmetic**, which simplifies the process of subtracting signed numbers. In standard subtraction, when subtracting two numbers (A - B), we would typically need a separate subtraction circuit. However, in **two's complement**, the subtraction can be reduced to **addition** of the first operand (A) and the **two's complement** of the second operand (B). This elegant property eliminates the need for a dedicated subtraction circuit, making it more efficient in hardware design.

In two's complement, the **most significant bit (MSB)** is used to represent the sign of the number:

- **MSB = 0** indicates a positive number.
- **MSB = 1** indicates a negative number.

For a 4-bit signed number in two's complement:

- The range of values spans from **-8 (1000)** to **+7 (0111)**. The two's complement representation of negative numbers is achieved by inverting all bits and adding 1 to the least significant bit (LSB).

Thus, subtraction of two signed numbers can be redefined as:

- **A - B** = **A + (-B)**, where **-B** is the two's complement of B.

This approach makes **signed subtraction** computationally simpler and more efficient, particularly in hardware implementations.

Design and Implementation in Verilog

The implementation of the **4-bit signed subtractor** involves the following steps:

- **Inputs**: Two 4-bit signed binary numbers, represented in two's complement format.
- **Operation**: To perform subtraction, the two's complement of the second operand (B) is computed. This is achieved by flipping the bits of B and adding 1. The result of this operation is then added to operand A.
- **Overflow Detection**: Since subtraction can potentially lead to overflow or underflow (when the result exceeds the representable range), we need to detect such conditions. The overflow occurs when both operands have the same sign but the result has an opposite sign.

4.3.2.1 Code

```
module Sign_Sub(O, A, B, S):
output reg[8:0]O;
input [3:0]A,B;
input [1:0]S;
// Function to perform 2's complement
function [8:0] Comp;
input [3:0] in;
begin 10
Comp = ~ {5b0000, in} + 1'bl; // Invert and add 1 (2's complement)
end
endfunction
always@(*) begin
case (S)
2'b00: O={5'b00000+A}-{5'00000+B}; //A-B
2'b01: O={5'b00000+A}-Comp(B);     //A-(-B)
2'b10: O=Comp(A)-{5'b00000+B};     //-A-B
2'b11: O=Comp(A)-Comp(B);          // (-A)-(-B)
endcase
end
endmodule
```

4.3.2.2 Simulation and RTL Viewer

From Figure 4.4 To ensure the correctness of the subtractor, a **testbench** is created to simulate various input conditions. These conditions include:

- **Positive - Positive**: Subtracting two positive signed numbers.
- **Negative - Negative**: Subtracting two negative signed numbers.
- **Positive - Negative**: Subtracting a negative number from a positive one.
- **Negative - Positive**: Subtracting a positive number from a negative one.
- **Boundary cases**: Subtracting edge values like **-8 (1000)** and **+7 (0111)** to test the overflow condition.

The simulation is performed using the **Xilinx Vivado** toolset, generating waveforms that show the correctness of the **subtraction** operation as well as the **overflow flag**. The results of the simulation confirm that the subtractor behaves correctly, handling both typical and edge cases as expected.

Fig 4.4 Simulation Output

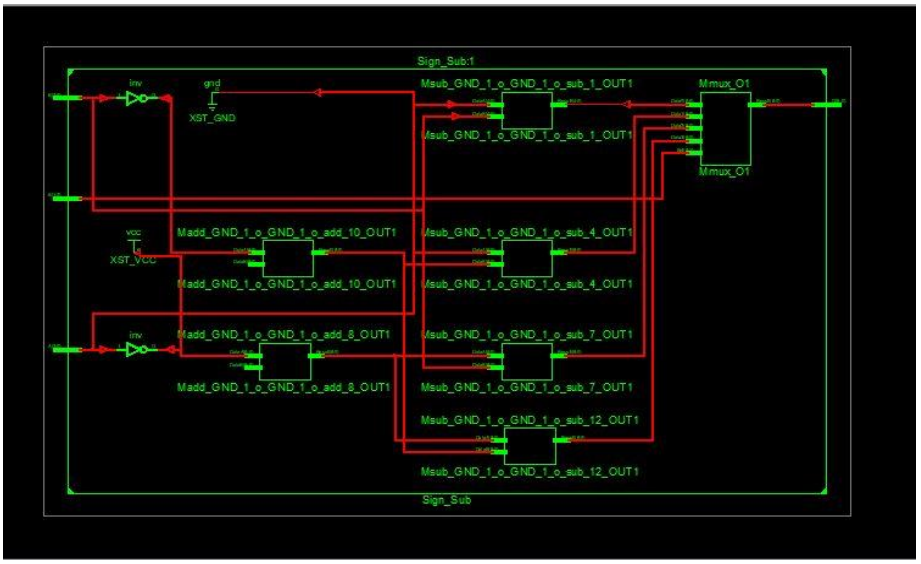


Fig 4.5 4-Bit Sign Subtractor RTL Viewer

4.3.3 4-Bit Sign Multiplier

The **4-bit signed multiplier** is a fundamental building block in arithmetic circuits, designed to multiply two signed 4-bit binary numbers represented in **two's complement format**. This project focuses on the complete design, implementation, and verification of a 4-bit signed multiplier using **Verilog Hardware Description Language (HDL)[26 27]**. The aim is to simulate the multiplier's behavior, validate its correctness through testbenches, and explore its integration on **FPGA hardware**, such as the **Xilinx Artix-7** platform. This project offers an insightful exploration into the world of **signed arithmetic** operations, bit-level logic design, and FPGA-based hardware development[2, 3].

Key Concept: Two's Complement Representation

In **two's complement**, a 4-bit signed number can represent values ranging from **-8** (1000) to **+7** (0111). The most significant bit (MSB) indicates the sign of the number, where:

- **MSB = 0** indicates a positive number.
- **MSB = 1** indicates a negative number.

Multiplication of signed binary numbers follows specific rules for both the **sign handling** and **overflow detection**. When multiplying two signed numbers, the **product** can be positive or negative, depending on the signs of the operands. This is determined by the following multiplication rules:

- Positive × Positive = Positive
- Negative × Negative = Positive
- Positive × Negative = Negative

Functional Overview and Design Approach

The design of the **4-bit signed multiplier** involves several steps:

- **Input Extension**: The two 4-bit signed numbers must be extended to 5 bits to account for the sign of each operand. The additional bit ensures that the sign of the input is properly handled during multiplication.
- **Binary Multiplication**: The core of the multiplier is the **binary multiplication** of two unsigned numbers. The inputs are treated as unsigned, and the partial products are

generated in the same manner as standard binary multiplication. However, for signed numbers, the **sign handling** is done separately.

- **Sign Handling**: After generating the partial products, the sign of the final product must be adjusted based on the signs of the operands:
    - If both operands are positive or both are negative, the product is positive.
    - If one operand is positive and the other is negative, the product is negative.
- **Two's Complement Adjustment**: When the product is negative, the result must be converted to **two's complement** format, which involves inverting the bits and adding 1 to the least significant bit.
- **Overflow Detection**: Overflow occurs if the product of two numbers exceeds the range that can be represented by the target bit width. In the case of a 4-bit signed multiplier, overflow occurs if the product is less than -64 or greater than +49. This condition must be detected and flagged.

4.3.3.1 Code

```
module Sign_Multiplier(O,A,B,S);
output reg[8:0]O;
input[3:0]A, B;
input[1:0]5;
// Function to perform 2's complement
function[8:0] Comp;
input[3:0] in;
begin
Comp = ~ {5b0000,in} + l'bl; // Invert and add 1 (2's complement)
end
endfunction
always@(*) begin
case (S)
2'b00: O={5'b00000, A}*{5'b00000, B};  //A*B
2'b01: O={5'b00000, A}*Comp(B);       //A*(-B)
2'b10: O=Comp(A)*{5'b00000, B};       // -A*B
2'b11: O=Comp(A)*Comp(B);            //(-A)*(-B)
endcase
end
```

endmodule

4.3.3.2 Simulation Output and RTL Viewer

Figure 4.6 Show the Output of the sign multiplier. Xilinx Vivado is used to simulate the design, and the resulting waveforms figure 4.7 confirm the accuracy of the multiplication and overflow detection logic.

- **Positive × Positive**: Testing when both operands are positive.
- **Negative × Negative**: Testing when both operands are negative.
- **Positive × Negative**: Testing when one operand is positive and the other is negative.

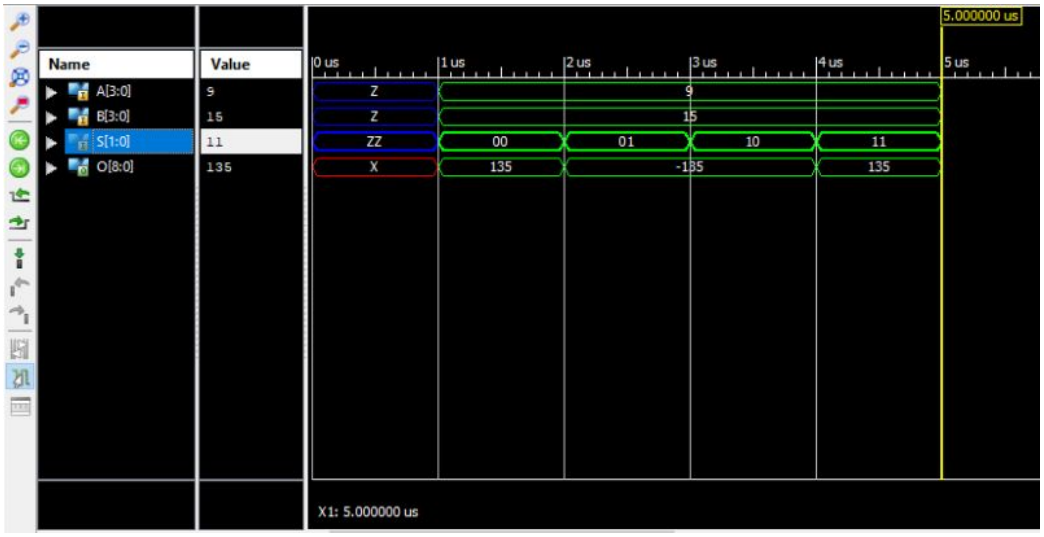


Fig 4.6 Simulation Output

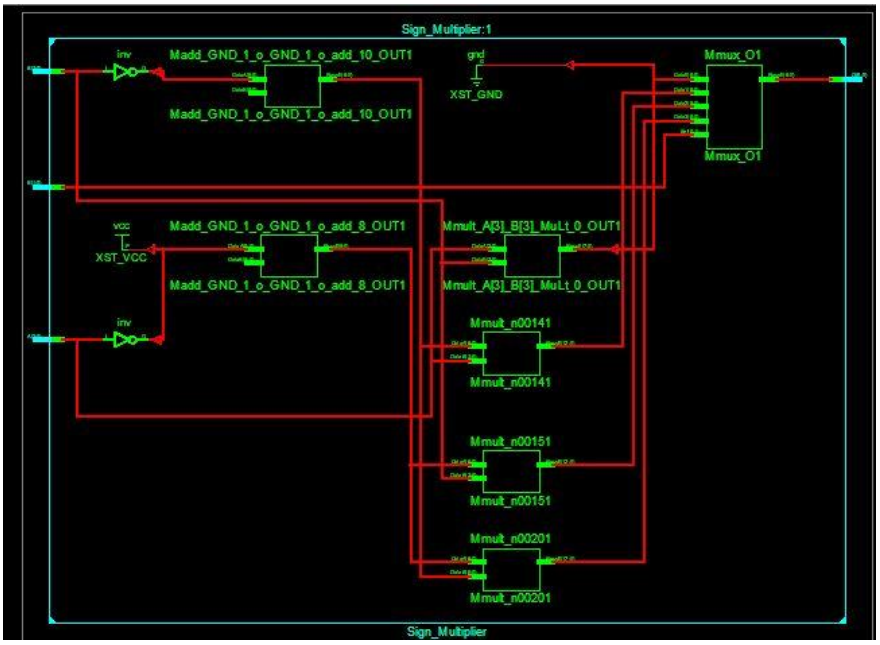


Fig 4.7 4-Bit Sign Multiplier RTL Viewer

4.3.4 4-Bit Sign Divider

The **4-bit signed divider** is a key arithmetic component in digital systems, specifically designed to perform division between two signed 4-bit numbers represented in **two's complement** format. This project focuses on the design, implementation, simulation, and verification of the signed divider using **Verilog Hardware Description Language (HDL)[1 27]**. The goal is to ensure the divider functions correctly in handling both positive and negative inputs, including handling exceptions such as division by zero and overflow conditions, and then deploy the design on an **FPGA platform**, specifically the **Xilinx Artix-7[1]**.

Key Concept: Two's Complement Representation

In **two's complement**, a 4-bit signed number can represent values ranging from **-8** (1000) to **+7** (0111). The most significant bit (MSB) serves as the sign bit:

- **MSB = 0** indicates a positive number.
- **MSB = 1** indicates a negative number.

The division operation must take into account the signs of the operands and perform the division accordingly. The signed divider should output both a **quotient** and a **remainder**, with appropriate sign corrections. Additionally, the divider must handle edge cases such as **division by zero** and **overflow**, which are critical for ensuring the robustness of the arithmetic unit in embedded systems.

Functional Overview and Design Approach

The 4-bit signed divider's functionality revolves around dividing one signed number (dividend) by another (divisor). The result of the division is a **quotient** and a **remainder** that are both signed, and the sign of the result is determined by the signs of the inputs.

Working Principle:

- **Sign Handling**: Before performing the division, the signs of both the **dividend** and the **divisor** are checked. If either operand is negative, their absolute values are used for the division, and the sign of the final result is determined by the **XOR** operation on the signs of the dividend and divisor.

- **Absolute Value Division**: The actual division operation uses the absolute values of the operands. In binary arithmetic, this can be implemented using a method like **shift-and-subtract**, where the divisor is subtracted from the dividend iteratively, with the quotient being incremented in each cycle.
- **Sign Adjustment**: After the division operation is completed, the **signs** of the quotient and remainder are adjusted:
    - If the dividend and divisor had the same sign, the quotient is positive; otherwise, it is negative.
    - The **remainder** always carries the same sign as the **dividend**, which is a key feature of signed division.
- **Overflow and Division by Zero**: The system must check for division by zero, as it is an invalid operation. Overflow occurs when the dividend is the smallest representable number (-8) and the divisor is -1, as this produces a quotient larger than the representable range.

4.3.4.1 Code

```
module Sign_divider (
input [3:0] A,B,   //4-bit signed input A, B
input [1:0] S,     // 2-bit select line for different operations
output reg [4:0] Q, R //4-bit signed remainder
);
// Function to perform 2's complement
function [8:0] Comp;
input [3:0] in;
begin
Comp =~{5'b00000, in} + 1'b1; // Invert and add 1 (2's complement)
end
endfunction
always (1) begin
if (B!=0) begin.
case (S)
2'b00: begin
Q= A/B;
R= A%B;
```

```
end
2'b01: begin
Q= Comp(A/B)-1'b1;
R=(A/B+1'bl)*B-A;
end
2'b10: begin
Q= Comp(A/B)-1'b1;
R=(A/B+1'bl)*B-A;
end
2'b11: begin
Q=A/B;
R=A%B:
end
endcase
end
else
begin
Q= 4'b0000;
R= 4'b0000;
end
end
endmodule
```

4.3.4.2 Simulation Output and RTL Viewer

Figure 4.8 Show the Output of the sign Divider. A testbench is developed to simulate the signed divider for various input combinations. The test cases include:

- Positive dividend and positive divisor (e.g., +8 ÷ +2)
- Positive dividend and negative divisor (e.g., +9 ÷ -3)
- Negative dividend and positive divisor (e.g., -6 ÷ +2)
- Negative dividend and negative divisor (e.g., -12 ÷ -4)
- Dividend is zero (e.g., 0 ÷ ±n)
- Divisor is one (e.g., ±n ÷ 1)

- Dividend equals divisor (e.g., ±n ÷ ±n)
- Overflow or boundary cases (e.g., minimum negative number ÷ -1)

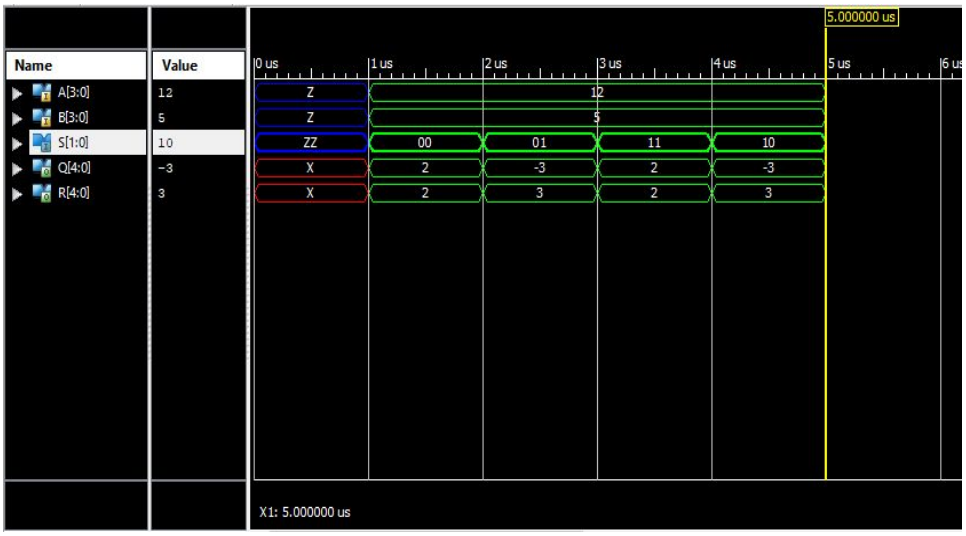


Fig 4.8 Simulation Output

Simulation is conducted using Xilinx Vivado, where the waveforms are analyzed to ensure that the results figure 4.9 are correct and that exceptions are properly handled. The divide-by-zero and overflow flags are checked to ensure that the module behaves as expected under edge cases

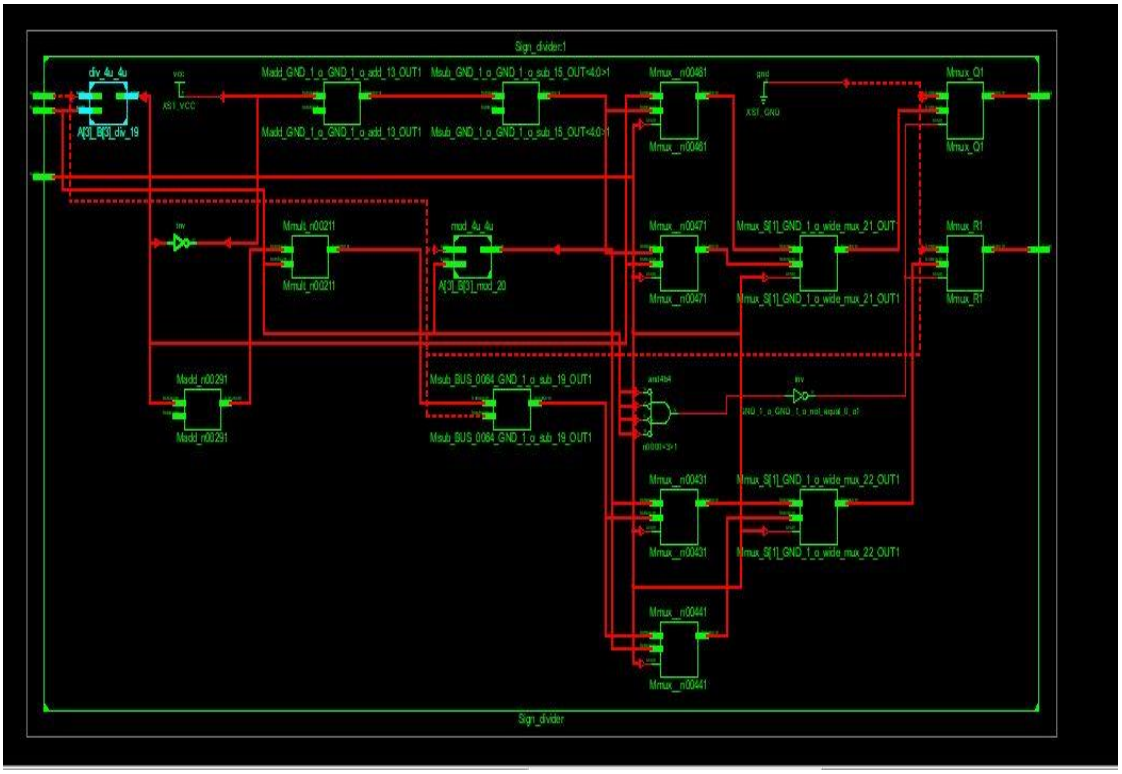


Fig 4.9 4-Bit Sign Divider RTL Viewer

4.4 **Hardware**: **Xilinx Artix-7 FPGA**

The **Xilinx Artix-7 FPGA** represents a highly optimized, cost-effective programmable logic device that caters to a wide range of applications in academic, industrial, and embedded systems. It is specifically designed for environments where high performance, power efficiency, and flexibility are paramount. Built on a **28nm process node**, the Artix-7 provides an ideal balance between **logic density**, **power consumption**, and **computational performance**. These features make it a suitable choice for various projects, especially those involving **digital signal processing (DSP)**, **high-speed computation**, and **embedded systems**.

For this project, titled "**Digital Sign Calculator: Verification and Implementation of Verilog HDL via FPGA Simulation**," the **Artix-7 FPGA** serves as the **hardware platform** where critical arithmetic operations such as **signed addition**, **subtraction**, **multiplication**, and **division** are synthesized, verified, and executed in real time. This process is essential for testing the functionality of digital designs and translating them from theoretical concepts into practical implementations.

Artix-7 FPGA Resources and Capabilities

The **Artix-7** family of FPGAs is built around the **7-series architecture** and comes with an array of advanced features that make it highly suited for high-performance applications. The Artix-7 provides a wide range of resources, including:

- **Configurable Logic Blocks (CLBs)**: These form the building blocks of the FPGA, allowing for complex digital designs to be implemented. The FPGA is equipped with thousands of CLBs, enabling it to handle intricate logic functions required for arithmetic operations such as those involved in the digital sign calculator.
- **DSP48E1 Slices**: These slices are designed to accelerate **multiplication** and **digital signal processing** tasks. The **DSP48E1** blocks are particularly valuable for implementing efficient multipliers, which are crucial for the multiplication and division operations of the digital sign calculator. These hardware multipliers allow the FPGA to perform these operations at much higher speeds than software implementations, making it ideal for real-time applications.
- **Block RAM**: The Artix-7 FPGA comes with dedicated **block RAM** that allows for fast data storage and retrieval. This is critical when performing operations on large datasets or for storing intermediate results during calculations. In the context of the digital sign

calculator project, block RAM can be used for temporary storage of operands, results, and flags (e.g., overflow or zero division).

- **Clock Resources**: The FPGA is equipped with a **100 MHz onboard clock**, which provides the timing reference needed for all operations. This clock ensures the synchronization of all modules within the FPGA, allowing for seamless execution of arithmetic operations in real time.
- **I/O Resources**: The Artix-7 also comes with various I/O resources, including **push buttons**, **switches**, and **seven-segment displays**. These components serve as the input and output interfaces for the digital sign calculator. Users can provide binary numbers via switches and visualize the results on the seven-segment display, enabling easy interaction with the FPGA and real-time testing.
- **Integrated Logic Analyzer (ILA)**: The Vivado Design Suite includes powerful debugging tools such as the **Integrated Logic Analyzer** (ILA), which allows for real-time monitoring of internal signals and performance during hardware execution.

Vivado Design Suite for Verilog HDL Development

The **Vivado Design Suite** from **Xilinx** is the primary software used for the design, simulation, synthesis, and implementation of the Verilog HDL code onto the FPGA. Vivado provides an integrated environment for the following steps in the design cycle:

- **Design Entry**: The project starts with writing Verilog HDL code that defines the behavior of the arithmetic operations (addition, subtraction, multiplication, and division) for signed numbers. The code is structured to handle two's complement representation, ensuring correct operation for both positive and negative inputs.
- **Simulation**: After the design is entered, the Vivado simulator is used to simulate the Verilog code. Simulation allows the designer to visualize the waveform outputs of the arithmetic operations, ensuring that the logic behaves as expected under different test conditions. This is a critical step for identifying potential issues before proceeding to synthesis and hardware deployment.
- **Synthesis**: In this phase, Vivado synthesizes the Verilog HDL code into a **netlist**, optimizing the logic for implementation on the Artix-7 FPGA. The synthesis process checks the design for resource usage, timing constraints, and overall efficiency.
- **Implementation**: Once the design is synthesized, the next step is **implementation**, where the netlist is mapped onto the specific resources available on the Artix-7 FPGA.

4.4.1 Hands-On Learning and Real-World Applications

The Artix-7 FPGA provides a hands-on environment for students, engineers, and researchers to transition from theoretical digital design to practical hardware implementations. By using the FPGA as a platform for implementing arithmetic operations such as signed addition, subtraction, multiplication, and division, users can bridge the gap between understanding Verilog HDL and working with real-world embedded systems.

The **Artix-7 FPGA**, with its combination of high-performance resources, **low power consumption**, and **flexible design capabilities**, is ideal for applications such as embedded processors, **digital signal processing** (DSP), and **system-on-chip (SoC)** designs. This makes it an invaluable tool for prototyping and testing **custom arithmetic units** and more complex **digital system**

Key Technical Features

Feature	Description
Architecture	28nm CMOS process with low static power consumption
Logic Cells	Up to 215,360 logic cells (depending on model)
DSP Slices	Integrated DSP48E1 slices for fast arithmetic operations
Block RAM	Up to 13,140 Kbits of distributed and block RAM
I/O Pins	Up to 500+ programmable I/Os with various voltage support
Clock Management	6 Clock Management Tiles (CMTs) with PLLs and MMCMs
High-Speed I/O	Supports 1.25Gbps to 6.6Gbps SerDes for high-speed data
Power Supply	Core voltage typically 1.0V or 1.2V, low power profile

Table 4.2 Key Feature

CHAPTER 5

PROPOSED METHODOLOGY

In modern digital systems, arithmetic operations are essential for various computational tasks, ranging from signal processing to embedded system control and real-time data analysis. The efficiency of these operations is critical, especially in resource-constrained environments where hardware usage must be minimized without sacrificing computational accuracy. For applications like embedded systems, sensor networks, or real-time data processing, achieving this balance can be quite challenging. This project focuses on the design, implementation, and verification of a **9-bit signed calculator** using **Verilog**, which is capable of performing basic arithmetic operations—such as **addition**, **subtraction**, and **comparison**—on signed integers within the range of **-256 to 255**. The calculator utilizes **two's complement representation** for signed numbers, ensuring it adheres to standard digital arithmetic conventions and maintains compatibility with conventional computational processes.

5.1 Motivation and Application

The motivation for this project stems from the increasing need for compact and efficient arithmetic units in embedded systems and other applications where hardware resources are limited. In these systems, especially those found in environments like automotive, IoT devices, or mobile systems, power consumption and physical space are constrained. A **9-bit signed calculator** strikes a balance between **simplicity** and **functionality** by providing a sufficient range of integers (from -256 to 255) for many real-world applications, without occupying excessive hardware resources. This makes it well-suited for tasks such as small-scale data processing, sensor fusion, or simple control algorithms, where more complex and larger bit-width systems might be overkill.

However, designing such a system brings with it several challenges, particularly in ensuring the correct and efficient handling of signed numbers. Signed numbers are commonly represented using **two's complement** format in digital systems. While this allows for easy arithmetic operations on both positive and negative values, the implementation must ensure proper management of the sign bit and correct overflow detection.

Another significant challenge is managing **overflow conditions**—when the result of an operation exceeds the allowed bit-width of the system. In the case of this 9-bit calculator,

operations that produce results beyond -256 or 255 must be flagged as overflow. Efficiently detecting overflow while ensuring that the design remains resource-constrained is essential for the correctness of the calculator.

5.2 Design Overview

To meet the requirements of this project, the proposed calculator design incorporates several key components, each working together to ensure accurate computation, effective overflow handling, and minimal resource usage:

- **Input and Output Registers**: The **input registers** are used to accept **signed 9-bit integers**, with two's complement representation, within the specified range. These registers are the interface through which values are fed into the system for processing. The **output registers** store the results of operations (addition, subtraction, comparison) before being output to the user or system.
- **Arithmetic Logic Unit (ALU)**: The **ALU** is at the heart of the calculator, performing the primary arithmetic operations: **addition**, **subtraction**, and **comparison**. For each operation, the ALU checks for overflow conditions and performs the operation using standard binary arithmetic techniques, taking care to adjust the result's sign based on the two's complement system.
- **Overflow Detection**: **Overflow** is a critical condition in signed arithmetic. In this 9-bit calculator, an overflow occurs when the result of an operation exceeds the range of -256 to 255. The system includes overflow detection logic that triggers an **overflow flag** whenever the result goes beyond the representable range. This mechanism is integrated within the ALU and ensures that users or systems interfacing with the calculator can handle overflow conditions appropriately.
- **Scaling of Inputs and Outputs**: While the calculator operates within the constraints of a 9-bit width, the design also incorporates scaling mechanisms. These scaling mechanisms allow the calculator to handle larger inputs and outputs during testing or in situations where the calculator is integrated into a larger system. For example, when connected to systems with broader data ranges, the calculator ensures that its outputs can be properly adjusted or scaled without causing errors.

5.3 Implementation and Simulation

The design of the 9-bit signed calculator is implemented in **Verilog HDL**, a hardware

description language widely used for modeling digital systems. The Verilog code defines the components of the calculator, including the input/output registers, the ALU, and the overflow detection logic. A **testbench** is written in Verilog to simulate the behavior of the calculator across various test cases. These test cases include standard operations like addition and subtraction, as well as edge cases that test the boundaries of the calculator's range, such as subtracting the minimum and maximum representable values or adding two numbers that result in overflow.

The simulation is run in **Xilinx Vivado**, a powerful suite for digital system design, simulation, and synthesis. Vivado provides a platform to simulate the Verilog design and visualize the results through waveform outputs. This simulation helps verify the correctness of the calculator's arithmetic operations, its handling of signed numbers, and its ability to detect overflow.

5.4 Efficiency and Resource Considerations

A significant focus of this project is ensuring the calculator's **efficiency**. In resource-constrained environments, particularly in embedded systems, the design must minimize the use of hardware resources such as logic gates, memory, and power consumption. The ALU and overflow detection logic are optimized to minimize gate usage while maintaining functional correctness.

The input/output registers are also designed to be compact and efficient, reducing the need for excessive control logic and ensuring that the overall system remains lightweight.

Input A	Input B	Sign Convention	Operations 2-bit	Results
10	10	00	Addition [00]	20
10	7	00	Subtractor [01]	3
10	8	00	Multiplier [10]	80
10	9	00	Divider [11]	1,1

Table 5.1 Function Table

The **Verilog HDL** design is modular and scalable, meaning that the components of the calculator (ALU, registers, overflow detection) can be easily adjusted or expanded for future designs. This scalability makes the calculator well-suited for integration into larger systems or for use in applications requiring more complex arithmetic functions.

Fig 5.1 Project Execution Plan

This project emphasizes the importance of optimization in digital design, focusing on minimizing resource utilization while ensuring robust performance. By implementing a compact signed calculator with efficient arithmetic processing capabilities, the project lays the groundwork for its integration into larger systems, such as digital signal processors or microcontrollers. The design also serves as a scalable framework for future enhancements, such as support for additional operations or increased bit-width.

CHAPTER 6

RESULTS AND OBSERVATIONS

The **Modulus-designed RTL viewer** (as shown in **Figure 3.4** of the appendix) represents a comprehensive design overview of the **Sign Calculator** system, which is implemented as a top-level module for performing basic arithmetic operations such as **addition**, **subtraction**, **multiplication**, and **division**. The **Sign Calculator** serves as the primary computational engine in this architecture, and the overall design features a well-organized structure that facilitates efficient arithmetic computation for signed numbers in two's complement format.

6.1 Module Structure and Inputs/Outputs

The blueprint of the design includes **fifteen outcome pins** and **fourteen input pins**, which allow data to flow in and out of the system. The **Syncro blocks** (synchronization blocks) are responsible for managing the flow of data based on the specific function chosen by the user, ensuring that the correct signals are directed into the **Sign Calculator** module. These synchronization blocks play a critical role in ensuring that the system behaves correctly by providing the necessary data for operations like addition, subtraction, multiplication, and division.

A notable feature of the design is the **syncro10 block**, which operates based on the number entered by the user. Once the user inputs the required number, this block helps ensure that the data is properly synchronized before being passed on to the main **Sign Calculator** module. This ensures that the data flow is managed seamlessly throughout the calculation process, preventing any data mismatches or timing issues.

The **Sign Calculator** module is responsible for executing all primary arithmetic operations. It is designed to handle signed 9-bit numbers and incorporates efficient logic to compute the result of various operations, including addition, subtraction, multiplication, and division. The module also plays an essential role in managing the **bit-width** of the registers and detecting any **overflow conditions** that may arise during operations.

6.2 Timing Performance and Optimization

The performance of the circuit is heavily dependent on the **timing** and **register management** within the design. As outlined in the **Report Timing** section of the circuit, the design's

performance has been thoroughly tested and optimized to meet specific timing constraints. The **timing performance** waveform summary displayed in **Figure 4** reveals key timing parameters that are critical for ensuring the circuit operates reliably within the specified limits.

The **Nexys A7 FPGA** was chosen as the platform for the implementation of the **Sign Calculator** blueprint. The Nexys A7 provides an ideal environment for real-time processing, with a **high number of registers** that support the efficient management of large amounts of data. The size and capacity of these registers directly impact the amount of data the system can handle at any given time, and thus, the FPGA's capability to perform real-time arithmetic calculations.

One of the key timing requirements is the **cycle time** of the system, which is set to **20.0 ns** per cycle. This is the fundamental time duration for each operation within the system, and ensuring that this cycle time is met is critical for achieving accurate and efficient computation. A critical timing check in the report is the **positive setup slack value**, which is reported to be **12.711 ns**. The setup slack indicates how much time is available between the **setup of data** at the input of the flip-flop and the clock edge that captures that data. A positive setup slack value indicates that the design successfully meets the timing requirement, ensuring reliable operation.

In addition to the setup slack, the **tactful delay** is another important performance parameter. In this design, the tactful delay is reported to be **4.985 ns**, which is well within the allowable limits. This delay ensures that the clock signal reaches the flip-flops in time to trigger the necessary operations.

Moreover, the **data delay**, which is the time required for a signal to travel from its source to the target flip-flop, is measured to be **5.758 ns**. This value represents the time it takes for the data signal to propagate through the circuit and reach the flip-flop that stores the result. The relatively short data delay indicates that the system is well-optimized for speed, and that the data reaches its target flip-flop well before the critical time threshold.

6.3 Register Management and Operation Flow

The **registers** in the system play a pivotal role in managing and storing the input and output data during each operation. **Register A** receives the **alternative input values** provided by the user. These values are processed based on the selected operation and subsequently sent to **Register B**, which serves as the storage location for the **final computed values**. After the

computation is complete, the resulting data is stored in **Register B**, from where it can be accessed or further processed.

As the operation proceeds, the **multiplexer** plays a crucial role in selecting the appropriate operation based on user input. For instance, the user may choose between different arithmetic operations, such as **multiplication**, **division**, or **subtraction**. The **multiplexer** responds to this user input by changing its state from **0 to 1**, which in turn dictates the operation that will be executed. This flexibility in operation selection allows the calculator to handle a variety of tasks, making it suitable for a wide range of applications.

The operations of multiplication, division, and subtraction are executed in a sequence, ensuring that each arithmetic task is handled correctly before moving on to the next. As the operations progress, the system continuously monitors for any **overflow conditions**. If an overflow occurs, the system triggers the appropriate flag to signal that the result is outside the permissible range. This ensures that any invalid results are detected and managed appropriately.

6.4 Overall System Performance

The **waveform illustration** shown in the report verifies that the timing constraints are met, ensuring that the data reaches the intended flip-flop within the required time limits. The design has been thoroughly tested, and the results confirm that it satisfies the necessary timing requirements.

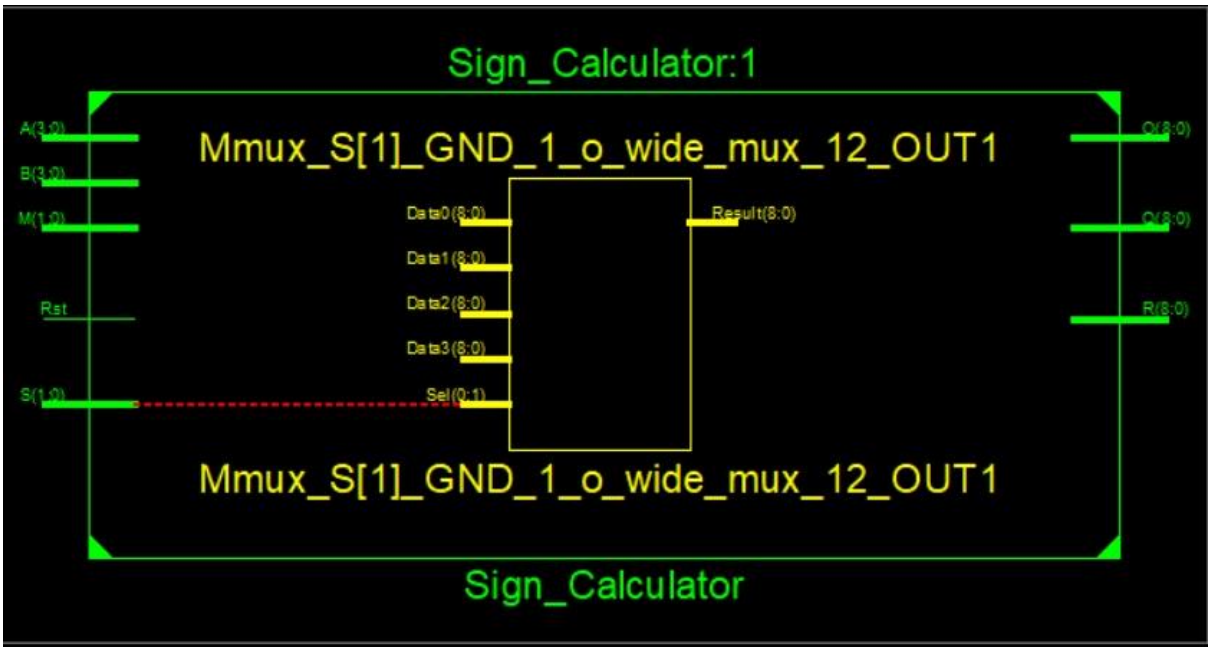


Fig 6.1 Sign_Calculator RTL Viewer

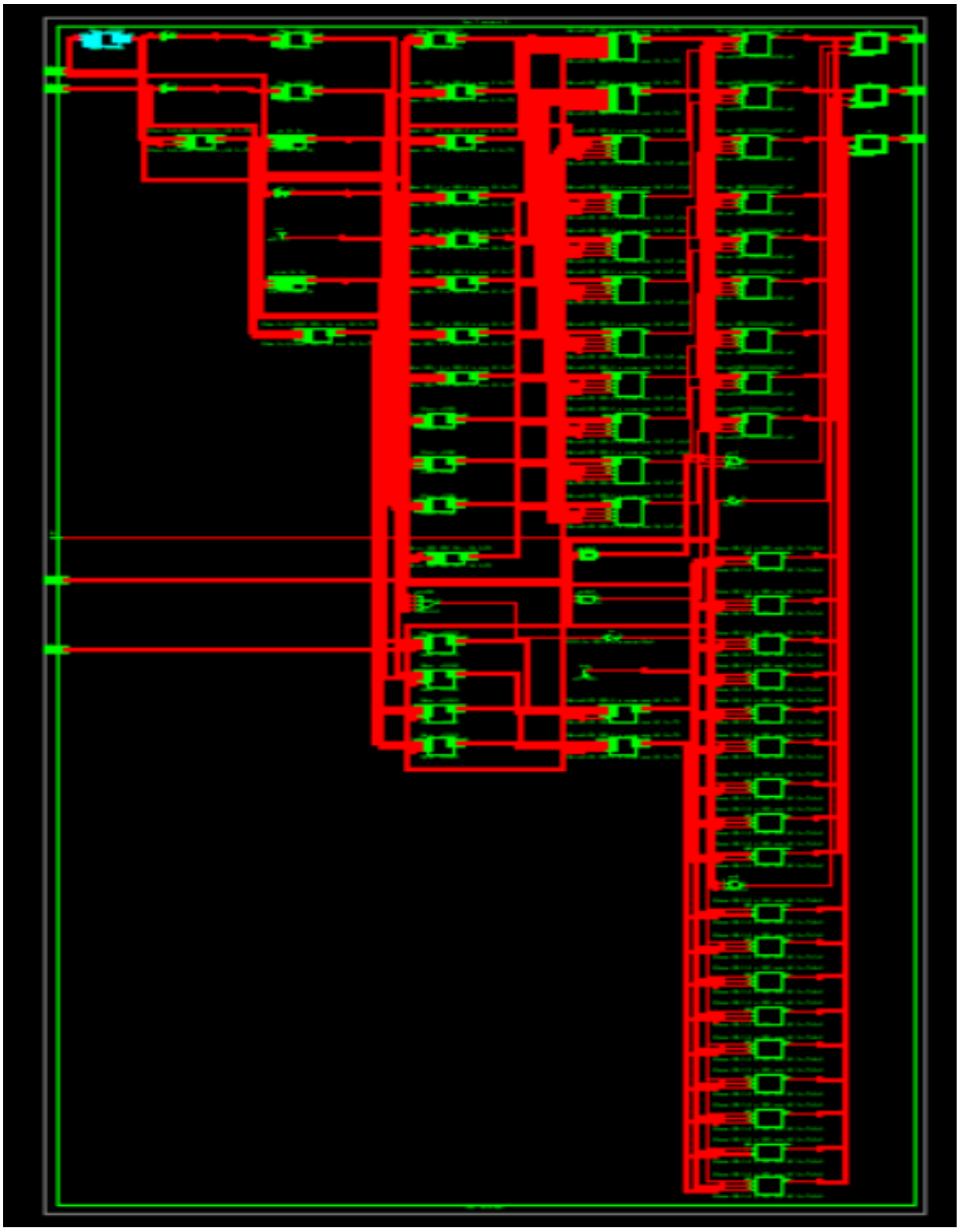


Fig 6.2 Top-level module RTL view ‘sign calculator’

Figure 6.2 illustrates the RTL (Register Transfer Level) view of the top-level module named ‘sigcalc’. This diagram represents the structural overview of how the digital logic is synthesized from the Verilog HDL code. The RTL schematic shows the interconnection of internal sub-modules, data flow paths, and signal routing within the design. It clearly highlights the input and output ports, arithmetic blocks, control logic, and data registers that collectively

form the architecture of the digital sign calculator. This graphical abstraction is crucial for understanding the logical hierarchy, verifying the correctness of module instantiation, and ensuring that the design meets the intended functionality before proceeding with simulation or hardware implementation.

6.4.1 CODE

```
module Sign_Calculator(O, R, Q, A, B, S, M,Rst);
output reg [8:0] O, R, Q;
input [3:0] A, B;
input [1:0] S, M;          // S: Sign control, M: Mode of operation
input Rst;
// Function to perform 2's complement
function [8:0] Comp;
input [3:0] in;
begin
Comp = ~{5'b00000, in} + 1'b1;  // Invert and add 1 (2's complement)
end
endfunction

// Function to perform Sign Adder
function [8:0] Sign_Adder;
input [3:0] A, B;
input [1:0] S;
begin
case(S)
2'b00: Sign_Adder = {5'b00000, A} + {5'b00000, B};     // A + B
2'b01: Sign_Adder = {5'b00000, A} + Comp(B);           // A + (-B)
2'b10: Sign_Adder = Comp(A) + {5'b00000, B};           // -A + B
2'b11: Sign_Adder = Comp(A) + Comp(B);                 // (-A) + (-B)
endcase
end
endfunction

// Function to perform Sign Subtractor
```

```
function [8:0] Sign_Sub;
input [3:0] A, B;
input [1:0] S;
begin
case(S)
2'b00: Sign_Sub = {5'b00000, A} - {5'b00000, B};      // A - B
2'b01: Sign_Sub = {5'b00000, A} - Comp(B);            // A - (-B)
2'b10: Sign_Sub = Comp(A) - {5'b00000, B};            // -A - B
2'b11: Sign_Sub = Comp(A) - Comp(B);                  // (-A) - (-B)
endcase
end
endfunction

// Function to perform Sign Multiplier
function [8:0] Sign_Multiplier;
input [3:0] A, B;
input [1:0] S;
begin
case(S)
2'b00: Sign_Multiplier = {5'b00000, A} * {5'b00000, B};      // A * B
2'b01: Sign_Multiplier = {5'b00000, A} * Comp(B);            // A * (-B)
2'b10: Sign_Multiplier = Comp(A) * {5'b00000, B};            // -A * B
2'b11: Sign_Multiplier = Comp(A) * Comp(B);                  // (-A) * (-B)
endcase
end
endfunction

// Function to perform Sign Divider
function [17:0] Sign_Divider;
input [3:0] A,B;      // 4-bit signed input A,B
input [1:0] S;        // 2-bit select line for different operations
reg signed [8:0] Q,R;
begin
if (B != 0) begin
```

```
case(S)
2'b00: begin
Q = A / B;
R = A % B;
end

2'b01: begin
Q = Comp(A / B) -1'b1;
R = (A/B+1'b1)*B - A;
end

2'b10: begin
Q = Comp(A / B)-1'b1;
R = (A/B+1'b1)*B - A;
end

2'b11: begin
Q = A / B;
R = A % B;
end
endcase
end else begin
Q = 9'b00000000;
R = 9'b00000000;
end
Sign_Divider = {Q, R};
end
endfunction

// Adittion, Subtraction, Multiplication, Divider Operatiom
always @(*) begin
if (Rst) begin
case(M)
2'b00: O = Sign_Adder(A, B, S);            // 00-Addition
```

```
2'b01: O = Sign_Sub(A, B, S);              // 01-Subtraction
2'b10: O = Sign_Multiplier(A, B, S);       // 10-Multiplication
2'b11: begin                               // 11-Divider
{Q,R} = Sign_Divider(A, B, S);
end
endcase
end else begin
O=9'b0;
Q=9'b0;
R=9'b0;
end
end
endmodule
```

Input A	Input B	Sign Convention	Operations 2-bit	Results
10	10	00	Addition [00]	20
10	7	00	Subtractor [01]	3
10	8	00	Multiplier [10]	80
10	9	00	Divider [11]	1,1

Table 6.1 Simulation Output

CHAPTER 7

CONCLUSION AND FUTURE WORK

The 9-Bit Sign Calculator project was undertaken with the objective of designing a digital system capable of performing arithmetic operations on 9-bit signed binary numbers. The project effectively demonstrates how signed binary arithmetic can be handled using hardware description languages like Verilog, emphasizing both the theoretical and practical aspects of digital design.

The calculator successfully performs addition and subtraction of 9-bit signed integers, represented in two's complement format. The choice of 9-bit width, as opposed to more conventional 8-bit or 16-bit systems, was made to demonstrate flexibility in design and to explore boundary conditions associated with odd bit-width architectures. In two's complement representation, the most significant bit (MSB) serves as the sign bit, where 0 denotes a positive number and 1 indicates a negative number. This design adheres to the standard signed arithmetic rules and ensures accurate interpretation of negative values during calculations.

In terms of implementation, the design was written in Verilog HDL and simulated using industry-standard tools such as ModelSim or Vivado. The simulations confirmed the correct operation of all implemented functions. The testbenches included a wide range of inputs covering normal operations, boundary values (like -256 and +255), and special cases such as addition of positive and negative numbers and subtraction leading to underflow or overflow conditions.

The modular design allowed for easier debugging and verification. Each module, such as the arithmetic unit, overflow detector, and sign evaluator, was individually tested before being integrated into the main system. Additionally, the project maintained good coding practices, including the use of parameterization, clean module hierarchies, and comprehensive comments, which enhanced the readability and reusability of the code.

This project has deepened our understanding of signed number representation, two's complement arithmetic, and Verilog-based digital design. It also provided hands-on experience in modeling complex logic circuits, simulating their behavior, and interpreting timing and functional simulation results.

This extension would also involve revisiting the overflow detection logic and the timing constraints, as larger bit widths may introduce propagation delays and require more efficient optimization techniques.

The **Digital Sign Calculator** has several practical applications in the field of digital electronics and embedded systems, particularly in environments that require **real-time arithmetic operations** with signed numbers. This project has practical applications across various fields, particularly in digital systems and embedded applications, where efficient arithmetic operations involving both positive and negative numbers are necessary within a constrained bit-width. One of the primary uses of the 9-bit sign calculator is in **embedded systems**, where it can perform essential arithmetic operations in real-time systems like robotics, signal processing, and sensor data interpretation. For example, in robotics, this calculator could be used to calculate parameters such as motor speed or position, which often involve signed values.

In **digital signal processing (DSP)** applications, where signed integers represent signals or coefficients, the 9-bit sign calculator plays a critical role in operations such as filtering, amplification, and transformations. Similarly, in **communication systems**, it can be used for modulation and demodulation tasks, error correction, and encoding/decoding processes. The 9-bit calculator is crucial in these applications because it handles negative and positive numbers correctly, ensuring accurate calculations for adjusting signal strength or phase.

The calculator is also useful in **industrial automation**, where signed integers represent physical quantities such as torque, force, or pressure. The 9-bit sign calculator can perform arithmetic operations on these values, such as when adjusting parameters or calculating metrics like force or load. In **financial systems**, it can compute simple operations such as profit or loss, where negative numbers represent losses and positive numbers represent gains.

Moreover, the **9-bit sign calculator** serves as a valuable **educational tool**, helping students and enthusiasts understand digital arithmetic concepts, such as two's complement representation, overflow detection, and the basics of number systems. It also has applications in **microprocessor design**, where it can be used in testing and building Arithmetic Logic Units (ALUs) or even custom processors that handle signed arithmetic. Overall, the 9-bit sign calculator is a versatile tool with applications ranging from control systems and industrial automation to educational projects and embedded designs.

REFERENCES

[1] Verilog HDL: An Introduction to Digital Design and Synthesis by S. Palnitkar. Pearson Learning.[2003]

[2] Digital Design: Principles and Practices by J. F. Wakerly. Pearson Prentice Hall.[2005]

[3] Kinney, L. L., & Roth, C. H. Designing Digital Systems with Verilog. Cengage Education.[2015]

[4] Digital Design with an Introduction to the Verilog HDL by Mano, M. M., and Ciletti, M. D. Pearson.[2017]

[5] Casio.http://www.vintagecalculators.com/html/casio.html is the access point. [2019].

[6] Vivado Design Suite: Synthesis, Xilinx Inc. https://www.xilinx.com is accessible.[2023]

[7] IEEE Standards Association, IEEE VHDL standard.[2005].

[8] ModelSim-Intel FPGA Edition User Guide, Intel Corporation. https://www.intel.com is the link.

[9] Xilinx Inc., Logic Simulation (UG900), Vivado Design Suite User Guide, 2023. [Online]. https://www.xilinx.com is accessible.

[10] Basys 3 Reference Manual, Digilent Inc. https://reference.digilentinc.com is the URL. [2022]

[11] All About Circuits, "Comprehending Binary Arithmetic and Two's Complement." https://www.allaboutcircuits.com is the link.

[12] "Verilog Code for Arithmetic Units and ALUs," FPGA4Student. https://www.fpga4student.com is the link.

[13] "Calculator's History."

[14] This is the URL: https://www.thecalculatorsite.com/articles/units/history-of-the-calculator.php. [2019].

[15] "Basic Logic Gates: A Digital Logic Gate Tutorial." https://www.electronicstutorials.ws/logic/logic_1.html is the link.[2019].

[16] "What Is the Process of a Calculator?"[Online].

[17] https://www.wonderopolis.org/wonder/how-does-a-calculator-work is what is accessible. [retrieved: November 20, 2019].

[18] Real world design with verilog "Link: https://www.learnabout-electronics.org/Digital/dig21.php" [2019].

[19] K.Coffman, FPGA design in the real world using Verilog. Prentice Hall PTR, Upper Saddle River, NJ, 2000.

[20] Brown, S. Digital Logic Fundamentals with Verilog Design. McGraw-Hill

Learning.[2013]

[21] Furman, M. D., and Hamblen, J. O. [2002]. Chapter 3 of Digital Systems Rapid Prototyping: A Guide (pp. 37-52). US: Springer, Boston, MA.

[22] Mathew, B., Stopyak, M., and Wagner, M. (2017) Final Project Paper for Simple Calculator ECE 2700-Digital Logic Design. [2017] ECE 2700: Digital Logic Design Final Project Paper: Simple Calculator.

[23] Colinares, D. S., Lousie, J. D., L., I. E., Q., E. R., & Caldo, R. B. (n.d.). Trigonometric operations are included in this 12-bit Verilog calculator. Trigonometric function 12-bit Verilog calculator, 3(No.), 43–48.

[24] Idris, N. R., Sutikno, T., Jidin, A. Z., and Jidin, A. (2012). Modified Non-Restoring Square Root Calculator with Simplified VHDL Coding. IJRES

[25] Shelke, M. G., Jaunjare, N. V., & Penshanwar, S. K. (2017). Calculator design with RISC (64 bit) architecture using VERILOG and FPGA. Calculator Design with RISC (64 Bit) Architecture Using VERILOG and FPGA.

[26] Yusmardiah, Y, Darmawaty, M., A, Abdul Karimi, H, Abdul Aziz, A., R, Ahmad, K., S. Translation of division algorithm into v h d l .translation of division algorithm into verilog HDL.[2010]

APPENDICES

Appendix A: Theoretical Background

The Digital Sign Calculator identifies whether a 9-bit binary number is positive or negative. In 2's complement representation:

- If the **most significant bit (MSB)** is 0, the number is **positive or zero**.
- If the **MSB is 1**, the number is **negative**.

For a 9-bit signed number:

- **Range**: −256 to +255
- **MSB (bit 8)** indicates the sign.

This module extracts the MSB to determine the sign and outputs:

- 0 → **Positive**
- 1 → **Negative**

This logic is implemented in **Verilog**, simulated, and deployed on an **FPGA** using switches and LEDs for I/O.

Appendix B: Verilog Code – Digital Sign Calculator

```
module Sign_Calculator(O, R, Q, A, B, S, M,Rst);
output reg [8:0] O, R, Q;
input [3:0] A, B;
input [1:0] S, M;          // S: Sign control, M: Mode of operation
input Rst;
// Function to perform 2's complement
function [8:0] Comp;
input [3:0] in;
begin
Comp = ~{5'b00000, in} + 1'b1;  // Invert and add 1 (2's complement)
end
endfunction

// Function to perform Sign Adder
function [8:0] Sign_Adder;
input [3:0] A, B;
input [1:0] S;
```

```
begin
case(S)
2'b00: Sign_Adder = {5'b00000, A} + {5'b00000, B};     // A + B
2'b01: Sign_Adder = {5'b00000, A} + Comp(B);           // A + (-B)
2'b10: Sign_Adder = Comp(A) + {5'b00000, B};           // -A + B
2'b11: Sign_Adder = Comp(A) + Comp(B);                 // (-A) + (-B)
endcase
end
endfunction

// Function to perform Sign Subtractor
function [8:0] Sign_Sub;
input [3:0] A, B;
input [1:0] S;
begin
case(S)
2'b00: Sign_Sub = {5'b00000, A} - {5'b00000, B};      // A - B
2'b01: Sign_Sub = {5'b00000, A} - Comp(B);            // A - (-B)
2'b10: Sign_Sub = Comp(A) - {5'b00000, B};            // -A - B
2'b11: Sign_Sub = Comp(A) - Comp(B);                  // (-A) - (-B)
endcase
end
endfunction

// Function to perform Sign Multiplier
function [8:0] Sign_Multiplier;
input [3:0] A, B;
input [1:0] S;
begin
case(S)
2'b00: Sign_Multiplier = {5'b00000, A} * {5'b00000, B};      // A * B
2'b01: Sign_Multiplier = {5'b00000, A} * Comp(B);            // A * (-B)
2'b10: Sign_Multiplier = Comp(A) * {5'b00000, B};            // -A * B
2'b11: Sign_Multiplier = Comp(A) * Comp(B);                  // (-A) * (-B)
```

```
endcase
end
endfunction

// Function to perform Sign Divider
function [17:0] Sign_Divider;
input [3:0] A,B;      // 4-bit signed input A,B
input [1:0] S;        // 2-bit select line for different operations
reg signed [8:0] Q,R;
begin
if (B != 0) begin
case(S)
2'b00: begin
Q = A / B;
R = A % B;
end

2'b01: begin
Q = Comp(A / B) -1'b1;
R = (A/B+1'b1)*B - A;
end

2'b10: begin
Q = Comp(A / B)-1'b1;
R = (A/B+1'b1)*B - A;
end

2'b11: begin
Q = A / B;
R = A % B;
end
endcase
end else begin
Q = 9'b00000000;
```

```
R = 9'b00000000;
end
Sign_Divider = {Q, R};
end
endfunction

// Adittion, Subtraction, Multiplication, Divider Operatiom
always @(*) begin
if (Rst) begin
case(M)
2'b00: O = Sign_Adder(A, B, S);           // 00-Addition
2'b01: O = Sign_Sub(A, B, S);             // 01-Subtraction
2'b10: O = Sign_Multiplier(A, B, S);      // 10-Multiplication
2'b11: begin                              // 11-Divider
{Q,R} = Sign_Divider(A, B, S);
end
endcase
end else begin
O=9'b0;
Q=9'b0;
R=9'b0;
end
end
endmodule
```

Appendix C: Simulation Output – Waveform

Tool Used: Xilinx ISE 14.0

Waveform Analysis:

- Shows transitions of A[8:0] input
- Output sign correctly reflects MSB (A[8])
- Simulation confirms correctness of logic

Appendix D: Raw Dataset Table

Input A	Input B	Sign Convention	Operations 2-bit	Results
10	10	00	Addition [00]	20
10	7	00	Subtractor [01]	3
10	8	00	Multiplier [10]	80
10	9	00	Divider [11]	1,1

Appendix E: FPGA Implementation Summary

- **Board Used**: Xilinx Nexyx A-7
- **Software**: ISE
- **Clock Frequency**: 50 MHz
- **I/O Mapping**:
    - A[8:0] → Switches SW0 to SW8
    - sign → LED0

Implementation Steps:

- Write Verilog Code
- Synthesize and Implement Design
- Perform Bitstream Generation
- Upload to FPGA via USB
- Use Switches to give input and observe LED output

Appendix F: Supporting Resources

- https://www.researchgate.net/publication/382370749_A_research_on_digital_signature_schemes
- https://www.nature.com/articles/s41598-024-83943-x
- https://www.youtube.com/watch?v=cWfaw7b3jKY
- https://www.youtube.com/watch?v=6ToR6vuRb3M
- https://www.youtube.com/watch?v=RKAQsyPRk_w&list=PLW38uDAjzio86mdWRaHXr_TB5GsD1XJa

RESEARCH PAPER

Volume-3, Issue-4, April 2025 International Journal of Modern Science and Research Technology
ISSN No- 2584-2706

Digital Sign Calculator Verification and Implementation of Verilog HDL via FPGA Simulation

Dr. Sarabjeet Kaur
Aditya Soraut
Amit Verma
Abhishek Kumar

Abstract:-
It is a possible to set up a digital sign calculator in the course of daily living. In this exploration composition, the pattern of a sign calculator is brought forward by using a format of Verilog HDL. Much synthetic Verilog code was created and obtained in Artix-7. The design of a 9-bit digital sign calculator can break fine mathematical operations similar to adder, subtract, multiplier, and divider. This digital sign calculator consists of nine-digit figures. The simulation procedure will be carried out in this article using devices from the Xilinx family. A shaft-style simulation of the calculator design is one of the outputs displayed in the waveform and RTL view. Calculator facts have been used favorably using Verilog HDL in relation to the behavior of the sign calculator functions.

Keywords-
Digital Sign Calculator, Circuit design, Verilog, Mathematical function

I. Introduction

The digital sign calculator consists of computer programs and hardware that can do basic arithmetic operations, such as adder, subtractor, multiplier, and dividor. In the scientific calculator there are numerous high-potential tasks, such as logarithmic, exponential, trigonometric, and hyperbolic functions, which might simplify challenging mathematical tasks.The development of calculators reduced the amount of time needed to explain complicated numbers and the mistakes that occur when calculating integers by hand. Scientific calculators included a number of extended operations, such as logarithmic, exponential, trigonometric, and hyperbolic Functions, that can break through complex mathematical functions. Binary format is also used for the calculators' input data. Additionally, the connected circuit will translate the decimal numbers into the base-two binary number system. Double data strings are used in integrated circuits to regulate transistors so that mathematical calculations may be performed. Mathematical tasks are completed, and the binary data is returned to the basic ten systems. The conclusion will show on the display screen. A complicated circuit is created by combining many logical gates to build adders. Numerous mathematical operations, including addition, subtraction, multiplication, division, and more, may be carried out by different combinations of logical gates inside the chips.

The creation of quicker, physically smaller designs with a greater number of gates and smaller sizes presents substantial problems for digital designers. To operate the device at the worst possible temperature, FPGA designers must produce designs that satisfy crucial requirements that other designers can comprehend. Furthermore, it is evident that the process variable circumstances may match the specifications and do not surpass the power consumption objectives. They are also reliable and verifiable. As a result, the designer writes and synthesizes hardware description language (HDL) code, which is then executed in the hardware chips. Verilog is crucial for business testing and replication of the outputs since it

DOI: https://doi.org/10.5281/zenodo.15273110

prevents mistakes. Free experiments are available in the Verilog simulation, and software tools for evaluating FPGA logic devices are also free. Hardware An overview Systems based on the chips with schematics have been developed using languages like Verilog HDL and VHDL. Signal logic is carried out by RTL, the design abstraction that controls the modeling of digital signals between hardware payrolls and synchronous digital circuits. Numerous studies on calculator design have been conducted, including RISC (64-bit) calculator design using Verilog HDL and FPGA Board; translation of splitting algorithms into Verilog programming language; FPGA prototype and a basic 9-bit calculator design bit slicing technique; updated non-restoring square root calculators with reduced VHDL code; and a novel approach for creating FPGA-integrated square root calculators.

II. The Flow Design and Data Type

The data types' block illustrations and the computational design processes are shown in Figure 1.Initially, the design was provided the input values; For data processing reasons, the structure is found in a decimal shape and transformed into a binary format. The digital sign calculator may be used for multiplication, division, addition, and subtraction, among other binary operations. The push-button portion of the synchronous unit has transformed the data from the asynchronous unit.A binary-to-decimal format is created for the unit operating at the 7-segment unit using the computed output data in binary format. The first result of the mathematical operations is created by indicating the number by illuminating different positions of the seven-segmented LEDs. These are nine sets of output LEDs with seven segments each. Eight of them display the number derived from the computation's outcome. The other, on the far left, displays the number indicator.

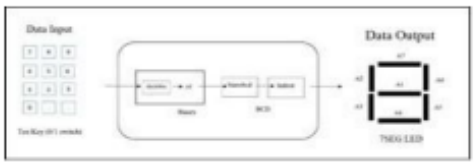


Figure 1. Flow and Data Type

The digital sign calculator's block depiction is seen in Figure 2. The investigation and advancement are performed utilizing Verilog HDL with the Xilinx ISE 14.0 computer program. In the Verilog HDL programming language, the calculator plan is divided into many significant corridors, each of which is programmed in a distinct module and activated using the RTL view and waveform lines. The "sign calculator" module is the best work module for the digital sign calculator in this business strategy.

All computing handling and capacities run in this module. As appeared in Figure 2,In the process of converting the input data from decimal to binary format, inputs from the "push" section were received in the blocks based on the block illustration. The "DECIMAL" function performs the information computation. This function stacks the twice supplied data to register A. Sign up The amount of memory consumed is returned by a function. For processing reasons, Register B stored the input number memory on the seven-segment LED screen. Implemented, reset (Rst), sign control(S), mode of operation (M), add, subtract, multiply, and divide are among the operators.

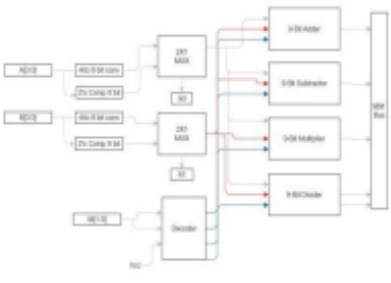


Figure-2. Sign Calculator block diagram

III. Bit Width Register

To ensure that the calculator's design can execute eight-digit computations, it is essential to evaluate the bit width range of the registers.The number of digits in a register is its bit width. Enroll A's inputs fall between 0 and 9999, meaning that in its two-fold version, they

DOI: https://doi.org/10.5281/zenodo.15273110

fall between 0 and 10111110101111000001111111. -199999998 (-99999999-99999999) to 199999998 (99999999+99999999) is the range of values in the decimal frame. When converted to its two-fold structure using the values of 199999998, it has 28 bits and is 10111110101111000001111111110. The formula for the sum of all bit ranges is 2 . Then let n be the representative of the number of bits, and $2^{\square} = 2^{28} = 268435456.2^{\square} = 2^{28} =$ 268435456. The binary representation of the decimal 268435456 is 10000000000000000000000000000. To arrange, the memory address has to be in the two-fold format.

IV. Overflow

When the number estimate exceeds the maximum bit area allowed in the registers, an overflow occurs. In order to prevent overflow, the articles require that entries be limited to a range between -999999999 and 999999999. The design overflow is calculated based on the decimal mode behavior. With the signed binary form of 0000_0101_1111_0101_1110_0000_1111_111 1, the input values must be greater than -999999999. Converting to its 2's complement yields the following value: 1111_1010_0000_1010_0001_1111_0000_000 1.

Therefore, when its 2's complement double value is translated back into decimal form, it becomes 4194967297. However, as this is the maximum value for a nine-digit calculator, the input values must not be greater than 999999999.

V. The Synchronous Circuit

Flip-Flop is made up of an edge detector circuit and a latch with an activation input. A signal clock serves as the edge locator's input. A square shaft with a fixed frequency is called a signal clock. The edge circuit for producing impulses during ascent is a positive edge-trigger flip-flop.

In the nine-digit calculator, flip-flops share a common reset and a signal clock, and an n-bit shift register is linked in series. This ensures accurate numerical operations by producing a counter where only one bit changes value between two consecutive counts. The offbeat flag "reset," which starts the count, is shared by all of the flip-flops.

D flip-flops make up the n-bit circuit; each flip-flop stores data on its own. These flip-flops can store and perform multi-digit computations efficiently since they are linked in parallel and share a reset and signal clock.

VI. Result And Discussion

The Modulus-designed RTL viewer is shown in Figure 4 at the higher value of the code in the appendix. "Sign calculator" is the name of the top-level module. There are fifteen outcome pins and fourteen input pins in the blueprint.

The syncro blocks operate based on the function selected by the user, directing the data into the 'sign calculator' module. In the meantime syncro10 will work based on the number entered by the user. Ultimately, all of the data enters the "sign calculator" module, which carries out the primary arithmetic operations, including addition, subtraction, multiplication, and division. This module also manages the bit width of registers and overflow detection.

The waveform summary in the Report Timing part of the circuit's timing performance is displayed in Figure 4. The Nexys A7 FPGA was used to implement the sign calculator blueprint. It enables effective data management and contains the most registers overall. The quantity of data the system can process at any given time depends on the size of these registers.

A time request for a cycle of 20.0 ns is provided by this circuit. With a positive setup slack value of 12.711 ns, the design satisfies the time requirements. The time required to extend the clock signal from the device pins to the target/source flip-flop's clock signal is known as the tactful delay, and it is 4.985 ns.

There is a 5.758 ns data delay. This illustrates how long it takes for a signal to get from its source to the intended flip-flop. The waveform illustration guarantees that the time restriction is fulfilled and that the data reaches the target before the required amount of time.

DOI: https://doi.org/10.5281/zenodo.15273110

Register A may receive the alternative input values. Register B is where the final values are entered. The calculated data is sent to register B. In order to process the result, Register B has stored the input number in memory. As with the expansion, let's say the customer selects the operation. In this instance, a high or active state is indicated by the multiplexer changing from 0 to 1. Division, multiplication, and subtraction are further operations.

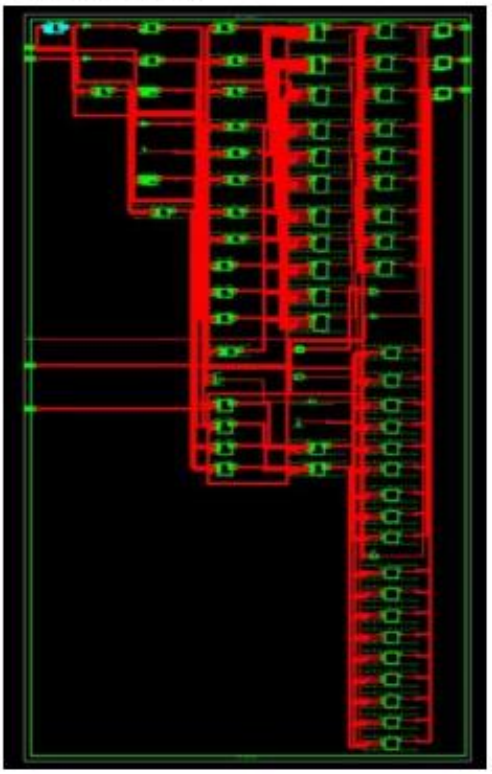


Figure-3. Top-level module RTL view 'sign calculator'

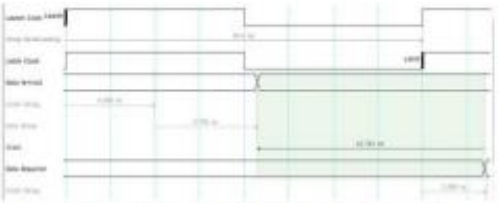


Figure-4. The Report Timing section of the circuit's waveform

An example of a waveform simulation with increment is shown in Figure 6. 20 ns per period was the timer's initial setting. The "equal" waveform was consistently raised to the highest possible level. The "reset" waveform was inserted into a d flip-flop type to manage the calculator design flux, depending on the proper circumstances. Pressing the double format value into a decimal waveform fits the desired values within the calculator's architecture. By placing their waveform in an advanced position, the desired chosen processes may be given names.

To enlist A, double input data is stacked into this function. The number of customers in the memory is returned by the Enlist A function. The first input number is taken by Enrol A. Subsequently, the input number that is substituted begins at 0. once the customer choose what they want. Figure 5 shows that 14 and 7 are the two inputs from the "decimal" signal that are stacked to enroll A. "Add," which stands for the addition operation, is the administrator selection. The calculated result yields at the "out" waveform and is stored in registers B and 21. This suggests that 21 is the consequence of adding the numbers 14 and 7. The system used for the final interpretation is the same for all other steps.

Figure-5. The Waveform of the 'add' function simulation.

The waveform reconstruction of the unique functions of the calculator design using Verilog HDL in this study, such as independent division, multiplication, and deduction, is shown in Figures 6, 7, and 8.

DOI: https://doi.org/10.5281/zenodo.15273110

Figure-6. The Waveform of the 'sub' function simulation

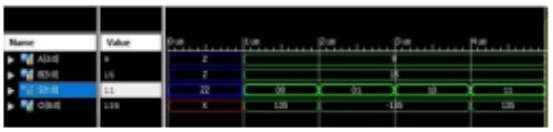


Figure-7. The Waveform of the 'mult function simulation

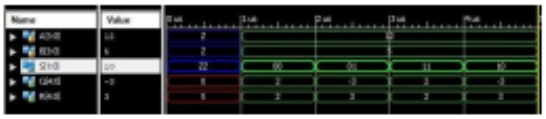


Figure-8 the Waveform of the 'div' function simulation

VII. Conclusion

This paper's goal is to plan the digital sign calculator's four integers and use the administrators in Verilog HDL. Verilog HDL programming was used to encode the calculator, and Xilinx ISE 14.0 was used to deconstruct it. In order for designers to create schematic chip frameworks that can be implemented into Field Programmable Gate Arrays (FPGAs), Verilog HDL is essential. By carrying out operations in a more advantageous manner than other languages, Verilog HDL appears to be able to represent and describe the self-assertive collection of computerized circuits. The device chosen from the Altera family of devices is the Nexys A7.

The calculator design has several limitations when it comes to constructing complicated digital circuits utilizing Verilog HDL because of comparable considerations as a lack of software and installations. Since the calculator's designs can only generate a 9-digit output and induce an integer answer, the outcome is not ideal if the answer contains decimal numbers. It is suggested that unborn work would produce a more accurate outcome; this is part of the floating-point process. Finally, it is advised to use more mathematical processes, such as logarithmic, to deconstruct intricate fine mathematical issues.

VIII. Confirmation

I would want to express my gratitude to Noida Institute of Engineering and Technology for giving me the space and chance to create this module.

IX. Source

[1] Vranesic, Z. & Brown, S. Digital Logic Fundamentals with Verilog Design. McGraw-Hill Learning.[2013]

[2] Verilog HDL: An Introduction to Digital Design and Synthesis by S. Palnitkar. Pearson Learning.[2003]

[3] Digital Design: Principles and Practices by J. F. Wakerly. Pearson Prentice Hall.[2005]

[4] Kinney, L. L., & Roth, C. H. Designing Digital Systems with Verilog. Cengage Education.[2015]

[5] Digital Design with an Introduction to the Verilog HDL by Mano, M. M., and Ciletti, M. D. Pearson.[2017]

[6] Casio.http://www.vintagecalculators.com/html/casio.html is the access point. [2019].

[7] Vivado Design Suite: Synthesis, Xilinx Inc. https://www.xilinx.com is accessible.[2023]

[8] IEEE Standards Association, IEEE VHDL standard.[2005].

[9] ModelSim-Intel FPGA Edition User Guide, Intel Corporation. https://www.intel.com is the link.

[10] Xilinx Inc., Logic Simulation (UG900), Vivado Design Suite User Guide, 2023. [Online]. https://www.xilinx.com is accessible.

[11] Basys 3 Reference Manual, Digilent Inc. https://reference.digilentinc.com is the URL. [2022]

[12] [12] All About Circuits, "Comprehending Binary Arithmetic and Two's Complement." https://www.allaboutcircuits.com is the link.

[13] "Verilog Code for Arithmetic Units and ALUs," FPGA4Student. https://www.fpga4student.com is the link.

DOI: https://doi.org/10.5281/zenodo.15273110

PLAGIARISM REPORT

PUBLICATIONS

Dr.Sarabjeet Kaur, Amit Verma, Aditya Soraut, Abhishek Kumar

International Journal for Modern Scientific Research and Technology (IJMSRT)

Volume: 11, Issue: 4, Month/Year: April 2025, ISSN: 2455-3778 (Online)

Partners

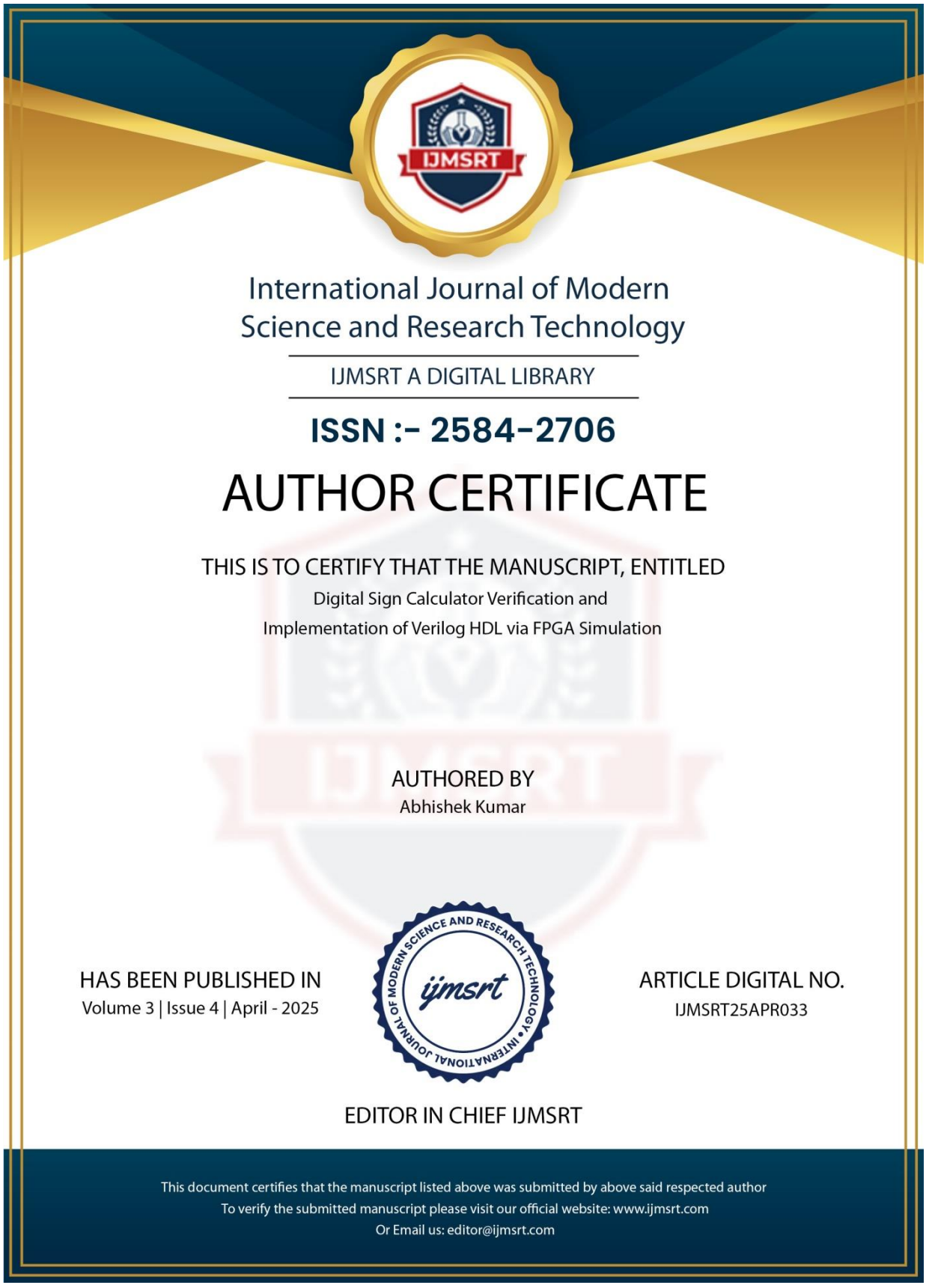
Google Scolar : https://tinyurl.com/bd9fm45c

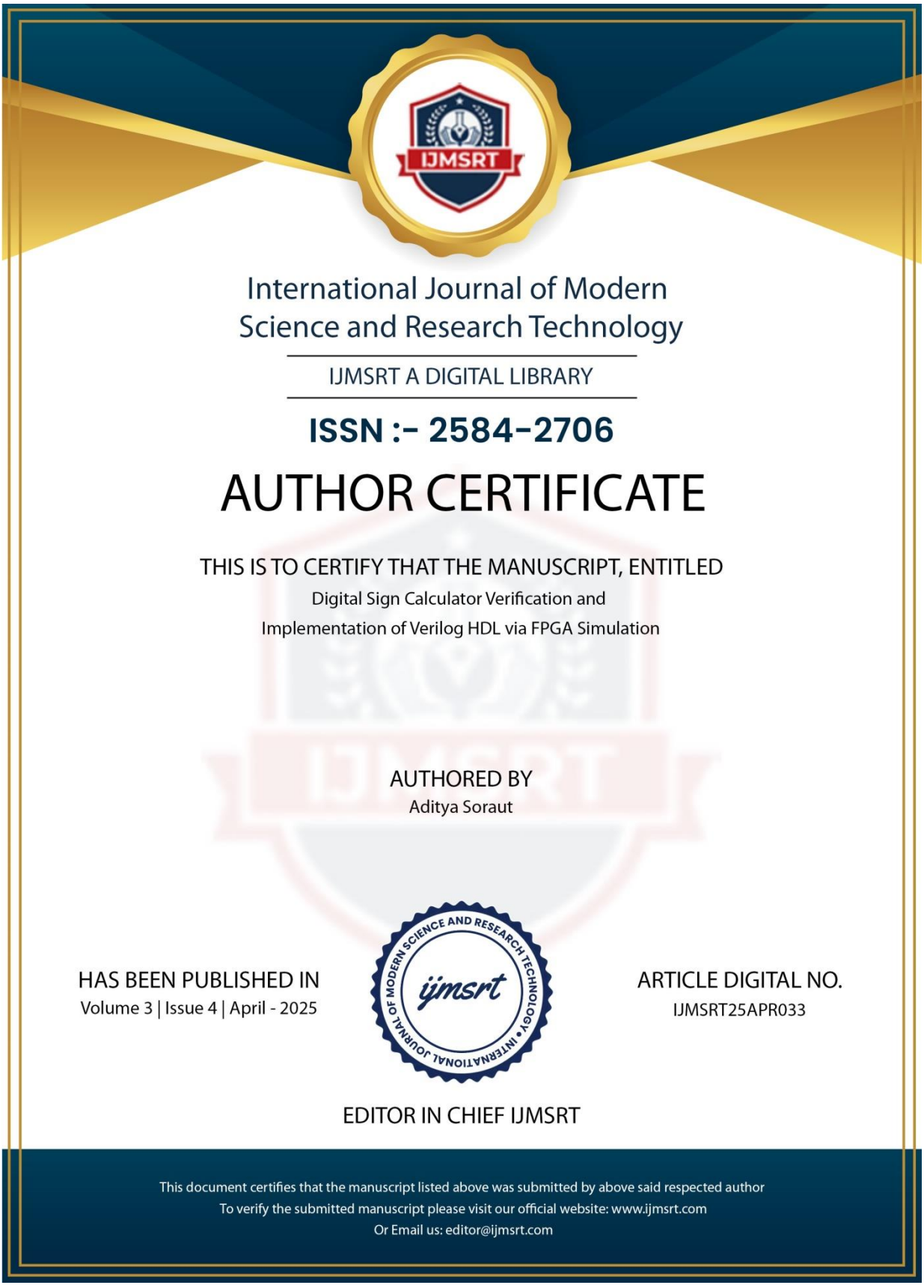
Scribd : https://tinyurl.com/2vp3zuhp

zenodo : https://doi.org/10.5281/zenodo.15273110

Publication Links: https://www.ijmsrt.com/articles/view/digital-sign-calculator-verification-and-implementation-of-verilog-hdl-via-fpga-simulation

AUTHOR CERTIFICATE





International Journal of Modern Science and Research Technology

IJMSRT A DIGITAL LIBRARY

ISSN :- 2584-2706

AUTHOR CERTIFICATE

THIS IS TO CERTIFY THAT THE MANUSCRIPT, ENTITLED

Digital Sign Calculator Verification and
Implementation of Verilog HDL via FPGA Simulation

AUTHORED BY

Aditya Soraut

HAS BEEN PUBLISHED IN

Volume 3 | Issue 4 | April - 2025

ARTICLE DIGITAL NO.

IJMSRT25APR033

EDITOR IN CHIEF IJMSRT

This document certifies that the manuscript listed above was submitted by above said respected author
To verify the submitted manuscript please visit our official website: www.ijmsrt.com
Or Email us: editor@ijmsrt.com

International Journal of Modern
Science and Research Technology

IJMSRT A DIGITAL LIBRARY

ISSN :- 2584-2706

AUTHOR CERTIFICATE

THIS IS TO CERTIFY THAT THE MANUSCRIPT, ENTITLED

Digital Sign Calculator Verification and
Implementation of Verilog HDL via FPGA Simulation

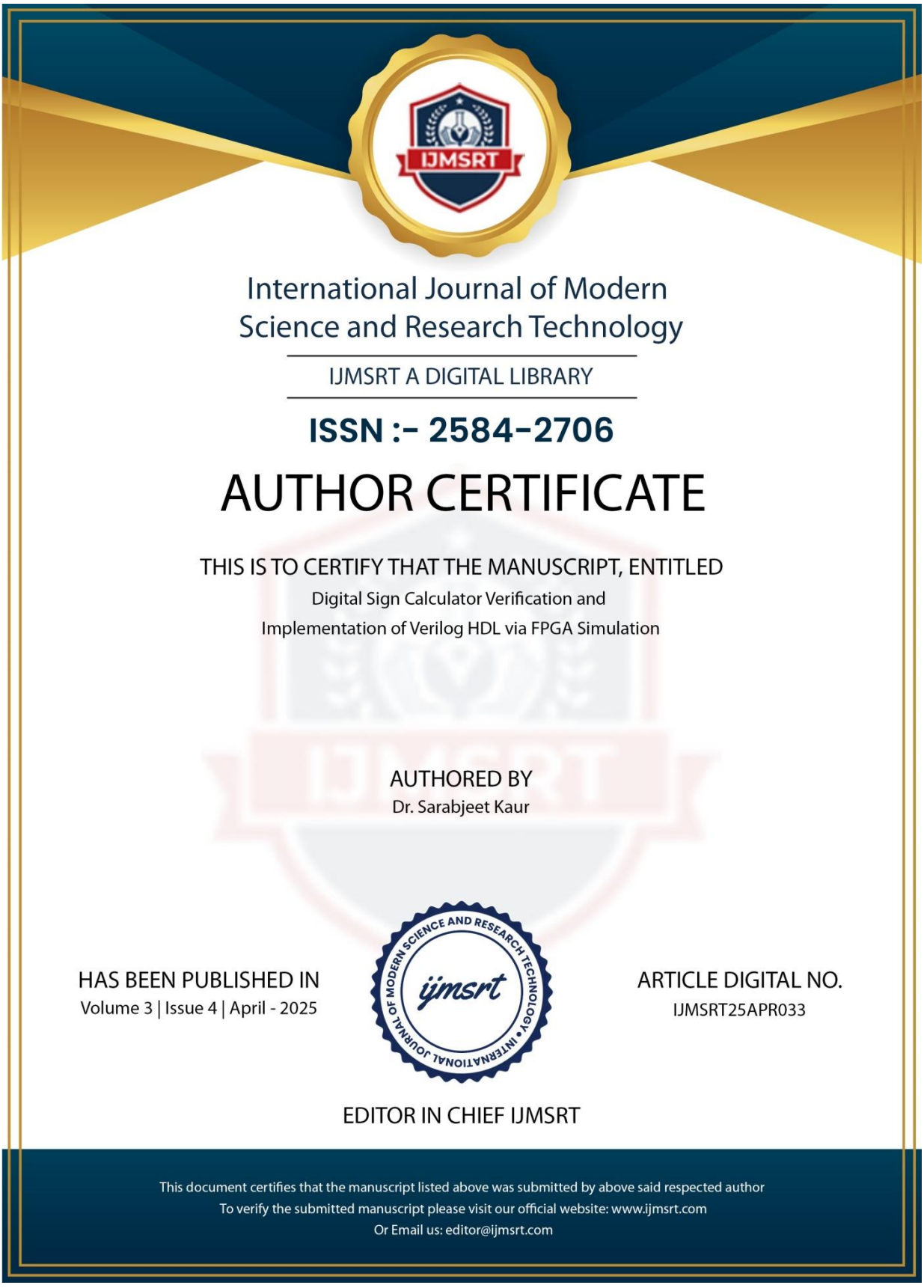
AUTHORED BY
Amit Verma

HAS BEEN PUBLISHED IN
Volume 3 | Issue 4 | April - 2025

ARTICLE DIGITAL NO.
IJMSRT25APR033

EDITOR IN CHIEF IJMSRT

This document certifies that the manuscript listed above was submitted by above said respected author
To verify the submitted manuscript please visit our official website: www.ijmsrt.com
Or Email us: editor@ijmsrt.com


IJMSRT
International Journal of Modern
Science and Research Technology
IJMSRT A DIGITAL LIBRARY
ISSN :- 2584-2706
AUTHOR CERTIFICATE
THIS IS TO CERTIFY THAT THE MANUSCRIPT, ENTITLED
Digital Sign Calculator Verification and
Implementation of Verilog HDL via FPGA Simulation
AUTHORED BY
Dr. Sarabjeet Kaur
HAS BEEN PUBLISHED IN
Volume 3 | Issue 4 | April - 2025
INTERNATIONAL JOURNAL OF MODERN SCIENCE AND RESEARCH TECHNOLOGY
ijmsrt
ARTICLE DIGITAL NO.
IJMSRT25APR033
EDITOR IN CHIEF IJMSRT
This document certifies that the manuscript listed above was submitted by above said respected author
To verify the submitted manuscript please visit our official website: www.ijmsrt.com
Or Email us: editor@ijmsrt.com

CURRICULUM VITAE

ABHISHEK KUMAR

ak182350@gmail.com
9798264687
Jitwaria, Samastipur, Bihar(848302)
https://www.linkedin.com/in/abhishek-kumar-5aa800232

OBJECTIVE

I'm a VLSI trainee, turning complex ideas into efficient silicon solution for modern electronics. Seeking an entry-level position in VLSI design and verification, where I can apply my knowledge of HDL programming, and circuit design. Eager to contribute to innovative chip development while continuously enhancing my technical skills in semiconductor technology.

SKILLS

- Verilog HDL
- Xilinx ISE (tool)
- Digital Electronics

EDUCATION

2021 - 2025	**Noida Institute Of Engineering and Technology** Bachelor of Technology(B.Tech)/Electronics and Communication Engineering 72.6%
2021	**St Paul Secondary School, Birsinghpur, Samastipur, Bihar** Higher Secondary(Class 12th) 72%
2019	**Campus Public School, Pusa, Samastipur, Bihar** Secondary (Class 10th) 68.8%

ACHIEVEMENTS & AWARDS

- Certified from Pine Training Academy for completion of training in Hardware digital design using HDL & Schematics and implementation on FPGA.

PROJECTS

- **Implementation and verification of Sign calculator using schematic & VHDL**
  The "Sign Calculator" project, implemented in Xilinx ISE using Schematics & Verilog/VHDL, determines whether an input number is positive, negative, or zero. The system uses combinational logic, represented by schematics. It demonstrates digital logic design, synthesis, and simulation on FPGA platforms.

- **FPGA Based traffic light controller**
  An FPGA-based system to control traffic lights at intersections, using finite state machines to manage light sequences for efficient traffic flow.

LANGUAGES

- Hindi
- English

PERSONAL INTEREST / HOBBIES

- Travelling

AMIT VERMA

+91-9625267756
AV553202@GMAIL.COM
http://www.linkedin.com/in/amit-verma-a60b41230
GHAZIABAD, UTTAR PRADESH

OBJECTIVE

Dedicated and results-driven electronics engineer looking for a role that challenges me to solve complex problems and improve existing electronic systems and products through the application of new technologies and techniques.

EDUCATION

2021-25	Noida Institute of Engineering and Technology
	B.Tech (Electronic and Communication), CGPA-7.72
2020	Dr Durgabai Deshmukh Memorial Senior Secondary School [CBSE, 70.6%]
2018	East Delhi Public School [CBSE, 71.2%]

SKILLS & ABILITIES

Xilinx ISE 14.7, Digital Circuit Design, Hardware Design, FPGA, C, C++, Python, DSA, System Verilog, Verilog, Hardware Description Language

EXPERIENCES/INTERNSHIP

From AUG To SEP 2023, I Worked as a SME at LEARNASYOUGO.

From SEP To OCT 2023, I Continued role as an SME at PHYSICS WALLAH.

From JUN 2024, Training from Pine Training Academy in VLSI & Embedded System.

PROJECTS

Hardware Design of 4-bit Sign Calculator and implementation on FPGAs by Schematic Circuit.

Hardware Design of 4-bit Sign Calculator and implementation on FPGAs by Verilog Code.

Self-Balancing Roboot by using Arduino nano.

Health Monitoring System by using Arduino uno.

LEADERSHIP/EXTRA CURRICULAR ACTIVITIES

Hosting CODE HUNT offline Coding Competitions for 250+ participants.

Created Coding Problem for the Competition.

Secure 2nd rank in NIET Sport Event (Cricket).

ADITYA SORAUT

36-Defence Colony 2 Mathura UP-281006

adityasoraut@gmail.com
9267022115
DOB 18/08/2002

Objective

Seeking to apply a diverse background in data analysis across various industries.Strong ability to discover and synthesize information and communicate findings clearly and concisely in support of business initiatives.

Experience

Physics wallah **04/03/2023 - present**
Subject matter expert
Clarify complex physics concepts and principles
Resolve doubts and misconceptions in physics theories and applications
Provide step-by-step explanations for physics problems and solutions

Myudates24.com **Aug/2024 - Present**
Information analyst intern
Collect, organize, and analyze data to identify trends and patterns
Develop and maintain databases, spreadsheets, and other data management tools

Education

Ramanlal shorawala public school
High School
2018 — 67.6

Subhash Inter College
Intermediate
2020 — 74.9

Noida institute of engineering and technology
Bachelor of Technology
2021-2025 — 8

Skills

- Data analysis
- Python
- Machine learning
- Team lead
- Effective communication
- Power Bi

Projects

A health recognition system IoT
It involves the use of Internet of Things (IoT) technology to collect and analyze health-related data from patients, enabling remote monitoring and early detection of health problems. The system typically consists of wearable devices or sensors that track vital signs, such as heart rate, blood pressure, and body temperature, and transmit the data to a central server or cloud-based platform for analysis.

Stock market prediction using machine learning

AC to DC Converter

Language

English , Hindi, German