Mini Project Report on

FPGA Based Calulator Using Logic Gates

Submitted by

Parag Anabhavane (Roll No. 01)
Aarya Bamne (Roll No. 04)
Diya Dave (Roll No. 09)
Sarang Desai (Roll No. 10)

in partial fulfillment for the award of the degree

TE(Semester-VI), Mini Project-2B

in

Department of Electronics & Telecommunication Engineering

under guidance of
Ms. Snehal Lopes

St. Francis Institute of Technology, Mumbai
Autonomous Institute Affilated to University of Mumbai
2024-2025

CERTIFICATE

This is to certify that Parag Anabhavane, Aarya Bamne,Diya Dave and Sarang Desai are the bonafide students of St. Francis Institute of Technology, Mumbai. They have successfully carried out the project titled "Basic Calculator using Basys 3" in partial fulfilment of the requirement for the award of Mini project-2B of third year (Semester-VI), in Electronics and Telecommunication Engineering of Mumbai University during the academic year 2024-2025. The work has not been presented elsewhere for the award of any other degree or diploma prior to this.

(Internal Examiner/ Reviewer 1)	**(External Examiner/ Reviewer 2)**
(Ms. Snehal Lopes) Name of Guide	**(Dr. Kevin Noronha) EXTC HOD**

(Dr. Sincy George)
Principal

ACKNOWLEDGEMENT

We are thankful to a number of individuals who have contributed towards our third year project and without their help; it would not have been possible. Firstly, we offer our sincere thanks to our project guide, Ms. Snehal Lopes for her constant and timely help and guidance throughout our preparation.

We are also grateful to the college authorities and the entire faculty for their support in providing us with the facilities required throughout this semester.

We are also highly grateful to Dr. Kevin Noronha, Head of Department (EXTC), Principal, Dr. Sincy George, and Director Bro. Shantilal Kujur for providing the facilities, a conducive environment and encouragement.

Signatures of all the students in the group

(Parag Anabhavane)

(Aarya Bamne)

(Diya Dave)

(Sarang Desai)

ABSTRACT

This project presents an FPGA-based calculator implemented on the Basys 3 development board using Verilog. The calculator performs basic arithmetic operations such as addition, subtraction, multiplication, and division. It leverages the FPGA's parallel processing capabilities to achieve high-speed computation with minimal latency. The user interacts with the calculator through input buttons or a connected peripheral, and the results are displayed on a seven-segment display or via UART communication. The design utilizes finite state machines (FSM) for operation control, along with combinational and sequential logic for efficient computation. This FPGA-based approach offers flexibility, low power consumption, and real-time responsiveness, making it suitable for embedded and high-speed computing applications..

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This project aims to develop an FPGA-based calculator using the Basys 3 development board, which features the Xilinx Artix-7 FPGA. The calculator will be capable of performing basic arithmetic operations such as addition, subtraction, multiplication, and division. Unlike software-based calculators, which rely on sequential execution, the FPGA-based approach enables faster computation through hardware-level parallelism. The design is implemented using Verilog and structured with finite state machines (FSMs) to control operations efficiently. User input is taken via switches, push buttons, or a serial interface, and the computed results are displayed on the onboard seven-segment display or communicated via UART. This project highlights the advantages of FPGA-based computation, demonstrating its potential for embedded systems and real-time applications.

1.1 Motivation

The motivation behind developing an FPGA-based calculator using the Basys 3 development board stems from the need to explore and demonstrate the advantages of hardware-based computation over software-based approaches. Unlike conventional calculators that rely on sequential processing, an FPGA-based design leverages combinational and sequential logic to execute arithmetic operations with minimal delay. This results in faster and more efficient computations, making it suitable for real-time applications.

1.2 Scope of Project

The FPGA-Based Calculator using the Basys 3 development board aims to implement a high-speed arithmetic computation system using hardware logic. The project covers various aspects of digital system design, including combinational and sequential logic, finite state machines (FSMs), and hardware description language (HDL) programming using Verilog. The FPGA-based calculator on Basys 3 demonstrates the efficiency, speed, and flexibility of hardware-based computation. It serves as an educational and practical tool for students, researchers, and engineers, with potential applications in embedded systems, industrial automation, and high-speed computing solutions.

1.3 Organization of Project

- **Literature survey**: Study of existing FPGA-based arithmetic units and calculators. Overview of Basys 3 features and capabilities. Exploration of Verilog-based digital design techniques. Comparison of different implementation approaches (combinational vs. sequential logic).

- **Hardware Used**: Hardware Components: Basys 3 development board, input/output peripherals. Functional Blocks: Arithmetic Logic Unit (ALU) for arithmetic operations. Input Interface (Switches, Buttons, UART). Output Interface (Seven-Segment Display, UART). Control Unit (Finite State Machine for operation handling). Block Diagram and Data Flow Representation.

- **Implementation using Verilog**: Development of individual Verilog modules for: Arithmetic operations (Addition, Subtraction, Multiplication, Division). User input handling. Control logic using FSM. Seven-segment display driver. UART communication module (if applicable). Integration of modules into a complete FPGA-based calculator system.

- **Simulation and Results**: Functional verification using Vivado Simulator. Testing of individual modules and full-system integration. Debugging and optimizing performance. Hardware testing on the Basys 3 FPGA board.

Chapter 2

Literature Survey

2.1 Literature Review:

This paper describes the implementation of a basic calculator using Verilog HDL, with simulation and testing conducted on an FPGA platform. The calculator was designed to perform four primary arithmetic operations—addition, subtraction, multiplication, and division. The authors focused on the design process, code structure, and simulation verification using FPGA tools like Quartus II. The work demonstrates how HDL can be used to model and test digital systems effectively before hardware deployment. It serves as a useful foundation for students and beginners learning FPGA development and digital logic design.

The study compares different 4-bit adder architectures—namely ripple carry, carry look-ahead, and carry select adders—implemented in conventional CMOS technology. The authors performed simulations to evaluate each architecture's performance in terms of propagation delay, power consumption, and area utilization. Their results show significant trade-offs among speed, complexity, and power efficiency, highlighting which designs are more suitable for specific applications. This analysis is particularly relevant for VLSI designers aiming to build optimized and efficient arithmetic units in embedded and low-power systems.

This work investigates the design and implementation of full subtractor circuits using various adiabatic logic techniques, such as Positive Feedback Adiabatic Logic (PFAL) and Efficient Charge Recovery Logic (ECRL). These techniques aim to reduce dynamic power loss by recovering and reusing energy during switching. The study compares these adiabatic designs to standard CMOS logic in terms of energy efficiency. The authors demonstrate that adiabatic subtractors can significantly lower power dissipation, making them ideal for use in low-power and portable electronic devices where energy conservation is critical.

This paper focuses on the FPGA-based implementation and analysis of different multiplication algorithms, including the shift-and-add method, Booth's algorithm, and the Wallace Tree multiplier. Each algorithm was synthesized and tested on an FPGA to assess performance in terms of speed, resource utilization (logic blocks), and computational efficiency. The results provide insight into the trade-offs between fast computation and hardware cost. The study is valuable for designers working on FPGA-based systems, such as digital signal processors (DSPs) and real-time embedded applications, where multiplication speed is a key performance factor.

Table 2.1: Summary of existing works on FPGA calculator using Logic gates

Author	Work Done	Remarks
Suhaili et al. (2020) [1]	Designed a calculator using Verilog HDL Simulated the design on FPGA.Performed basic arithmetic operations. .	Good for learning FPGA and Verilog basics.
Sarker et al. (2024) [2]	Compared different 4-bit adder architectures.Analyzed power, delay, and area in CMOS.	Helpful for selecting efficient adders
Sanadhya and Sharma (2020) [3]	Designed full subtractors using adiabatic logic. Compared them with CMOS designs.	Focuses on low-power design techniques.
Agrawal and Patel (2017) [4]	Implemented multiple multiplication algorithms on FPGA.Compared speed and resource usage.	Practical for real FPGA projects.

Chapter 3

Software Used

3.1 Vivado

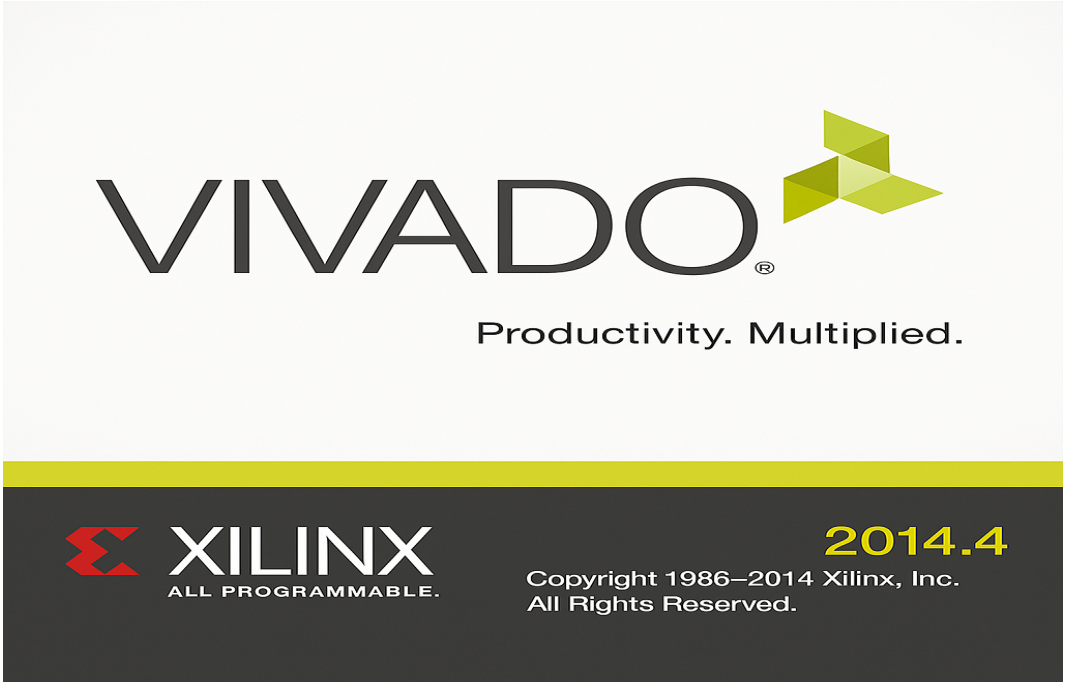

Figure 3.1: Vivado Software.

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of hardware description language (HDL) designs, superseding Xilinx ISE with additional features for system on a chip development and high-level synthesis. Vivado represents a ground-up rewrite and re-thinking of the entire design flow (compared to ISE).Like the later versions of ISE, Vivado includes the in-built logic simulator. Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.Vivado was introduced in April 2012, and is an integrated design environment (IDE) with system-to-IC level tools built on a shared scalable data model and a common debug environment. Vivado includes electronic system level (ESL) design tools for synthesizing and verifying C-based algorithmic IP; standards based packaging of both algorithmic and RTL IP for reuse; standards based IP stitching and systems integration of all types of system building blocks; and the verification of blocks and systems.

3.1.1 Cicuit Simulation Software

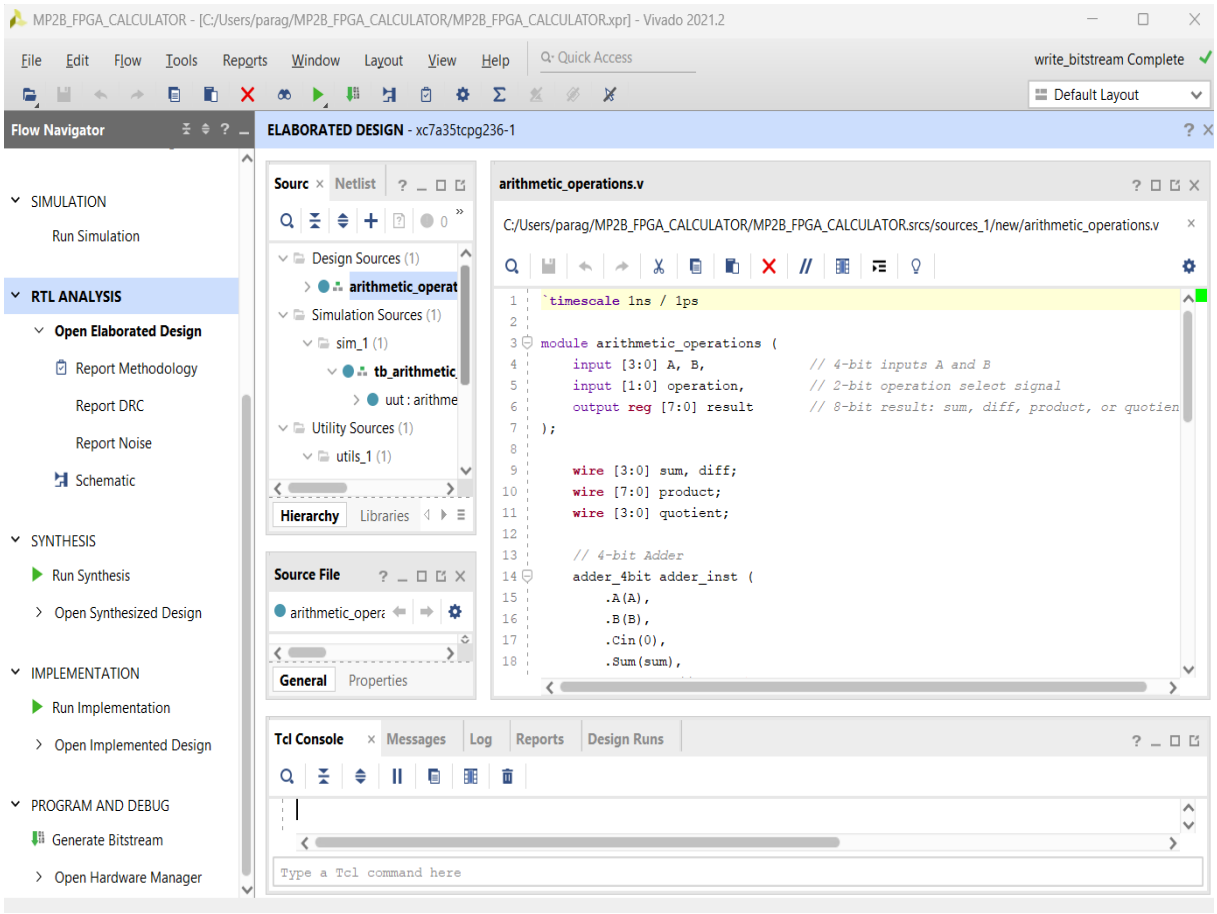

Figure 3.2: Vivado Software.

This Circuit simulation software is a powerful tool used in electronics and electrical engineering to design, test, and analyze circuits without the need for physical prototypes. It allows engineers to simulate the behavior of both analog and digital circuits under various conditions, helping to identify design flaws early in the development process. These tools typically include schematic capture for drawing circuits, simulation engines for analyzing behavior, and waveform viewers for visualizing signals. A prime example is Vivado Design Suite by Xilinx, which is widely used for FPGA development. Vivado allows users to write and simulate hardware description languages like Verilog or VHDL, as seen in the current project where an arithmetic operations module is being developed and tested. The software provides capabilities such as RTL analysis, logic synthesis, implementation, and simulation of digital systems, making it ideal for designing and verifying complex digital circuits. Overall, circuit simulation software greatly enhances efficiency, accuracy, and reliability in electronic system design.

Chapter 4

Hardware Used

4.1 Basys 3

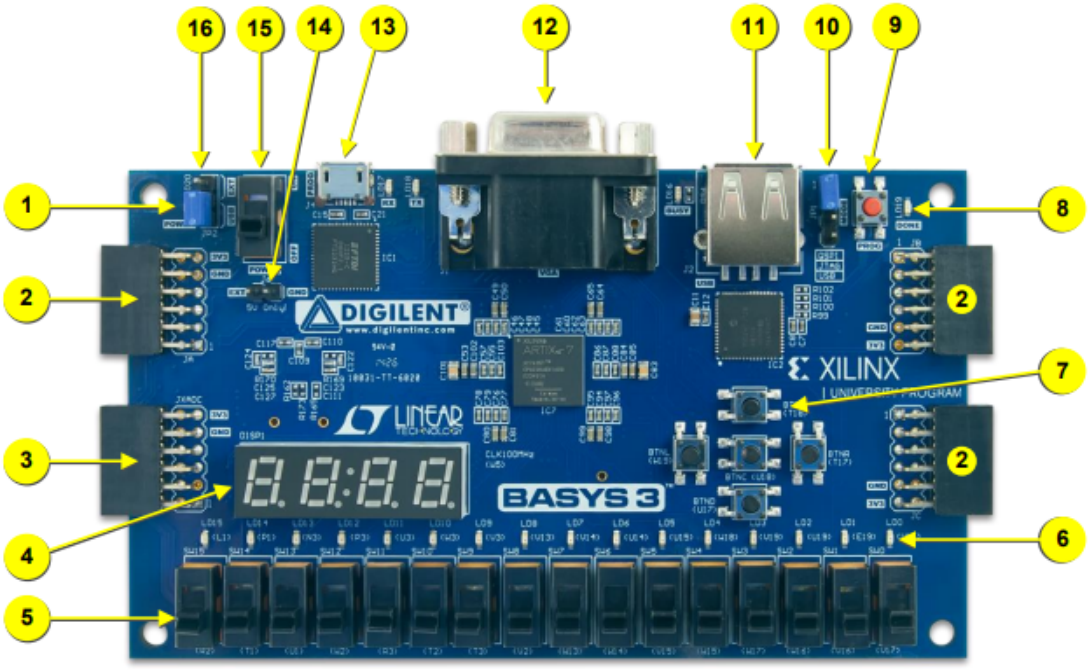

Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure 4.1: Basys 3

-

The Basys 3 is a powerful and flexible FPGA development board designed by Digilent and based on the Xilinx Artix-7 FPGA (XC7A35T-1CPG236C). It is ideally suited for beginners, students, and hobbyists who want to design and implement digital systems. The board is equipped with various built-in features that eliminate the need for external components, making it an all-in-one platform for learning and development. It supports both JTAG and USB configuration, with a Power Good LED (Callout 1) that indicates the board is properly powered. The FPGA Programming Done LED (Callout 8) lights up once the FPGA is successfully programmed. For interaction and debugging, the board includes 16 slide switches (Callout 5), 16 LEDs (Callout 6), and 5 pushbuttons (Callout 7), which allow for real-time input/output control and observation.

To enhance interfacing and visualization capabilities, the Basys 3 includes a Four-digit 7-segment display (Callout 4), ideal for displaying decimal or hexadecimal values. It also features a VGA connector (Callout 12), which allows simple graphical output to a monitor for visualization of digital outputs or custom graphics. The board contains Pmod ports (Callout 2) for modular device connections and an Analog signal Pmod port/XADC (Callout 3) for analog input via the FPGA's onboard ADC. For serial communication and programming, the board features a USB host connector (Callout 11), UART/JTAG USB port (Callout 13), and Programming mode jumper (Callout 10) for switching between JTAG and Quad SPI programming. An FPGA reset button (Callout 9) allows users to manually reset the configuration.

The Basys 3 board can be powered either through a USB port or an external power connector (Callout 14). Power selection is managed using the Power Select Jumper (Callout 16), and the entire board can be toggled ON/OFF using the Power Switch (Callout 15). This flexible power management ensures the board can operate effectively in low and high-power projects. With these features and its compact layout, the Basys 3 serves as a complete platform for implementing mini-projects like digital calculators, counters, traffic light controllers, and communication protocols. Its robust design and in-built peripherals make it an excellent choice for learning digital design using Verilog or VHDL in tools like Xilinx Vivado.

The Basys 3 board is primarily programmed using Xilinx Vivado Design Suite, which provides a robust environment for writing HDL code (such as Verilog or VHDL), simulating digital designs, synthesizing logic, and generating bitstreams for FPGA configuration. Students and developers can design combinational and sequential logic circuits, implement finite state machines, and integrate real-time input/output controls with onboard peripherals. The board's flexibility and built-in features make it suitable for a wide range of applications including digital calculators, traffic light controllers, ALU designs, serial communication interfaces (UART/SPI), real-time counters, and display-based projects. Additionally, the Pmod ports allow further expansion with modules like temperature sensors, Bluetooth, motor controllers, or camera interfaces, making Basys 3 an excellent platform for both academic learning and advanced mini-project implementations in embedded systems and digital electronics.

Chapter 5

Coding

5.1 Instance Verilog Code

```
`timescale 1ns / 1ps
module arithmetic_operations (
    input [3:0] A, B,           // 4-bit inputs A and B
    input [1:0] operation,      // 2-bit operation select signal
    output reg [7:0] result     // 8-bit result: sum, diff, product, or quotient
);
    wire [3:0] sum, diff;
    wire [7:0] product;
    wire [3:0] quotient;

    // 4-bit Adder
    adder_4bit adder_inst (
        .A(A),
        .B(B),
        .Cin(0),
        .Sum(sum),
    );

    // 4-bit Subtractor
    subtractor_4bit subtractor_inst (
        .A(A),
        .B(B),
        .Diff(diff),
    );

    // 4-bit Multiplier
    mult4bit multiplier_inst (
        .a(A),
        .b(B),
        .p(product)
    );
    // 4-bit Divider (Quotient only)
    divider_4bit divider_inst (
        .A(A),
        .B(B),
        .Quotient(quotient)
    );
    always @(*) begin
        case (operation)
            2'b00: result = {4'b0, sum};       // Addition
            2'b01: result = {4'b0, diff};      // Subtraction
            2'b10: result = product;           // Multiplication
            2'b11: result = {4'b0, quotient};  // Division
            default: result = 8'b0;
        endcase
    end
endmodule
```

Figure 5.1: arithmetic operations verilog combined code

5.2 Full Adder

```
`timescale 1ns / 1ps

module adder_4bit (
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout
);
    wire c1, c2, c3;

    full_adder FA0 (.a(A[0]), .b(B[0]), .cin(Cin), .sum(Sum[0]), .cout(c1));
    full_adder FA1 (.a(A[1]), .b(B[1]), .cin(c1), .sum(Sum[1]), .cout(c2));
    full_adder FA2 (.a(A[2]), .b(B[2]), .cin(c2), .sum(Sum[2]), .cout(c3));
    full_adder FA3 (.a(A[3]), .b(B[3]), .cin(c3), .sum(Sum[3]), .cout(Cout));
endmodule

module full_adder (
    input a, b, cin,
    output sum, cout
);
    wire w1, w2, w3;

    xor (w1, a, b);
    xor (sum, w1, cin);
    and (w2, a, b);
    and (w3, w1, cin);
    or (cout, w2, w3);
endmodule
```

Figure 5.2: Full Adder

5.3 Full Subtractor

```
`timescale 1ns / 1ps

module subtractor_4bit (
    input [3:0] A, B,
    output [3:0] Diff,
    output Borrow
);
    wire b1, b2, b3;

    full_subtractor FS0 (.a(A[0]), .b(B[0]), .bin(1'b0), .diff(Diff[0]), .bout(b1));
    full_subtractor FS1 (.a(A[1]), .b(B[1]), .bin(b1), .diff(Diff[1]), .bout(b2));
    full_subtractor FS2 (.a(A[2]), .b(B[2]), .bin(b2), .diff(Diff[2]), .bout(b3));
    full_subtractor FS3 (.a(A[3]), .b(B[3]), .bin(b3), .diff(Diff[3]), .bout(Borrow));
endmodule

module full_subtractor (
    input a, b, bin,
    output diff, bout
);
    wire w1, w2, w3;

    xor (w1, a, b);
    xor (diff, w1, bin);
    and (w2, ~a, b);
    and (w3, ~w1, bin);
    or (bout, w2, w3);
endmodule
```

Figure 5.3: Full Subtractor

5.4 4-Bit Multiplication using shift-and-add method

```
`timescale 1ns / 1ps
module mult4bit(
    input [3:0] a,
    input [3:0] b,
    output [7:0] p
);

    wire [3:0] m0, m1, m2, m3;
    wire [4:0] m1_shifted;
    wire [5:0] m2_shifted;
    wire [6:0] m3_shifted;
    wire [7:0] s1, s2, s3;

    // Generate partial products
    assign m0 = {4{a[0]}} & b;
    assign m1 = {4{a[1]}} & b;
    assign m2 = {4{a[2]}} & b;
    assign m3 = {4{a[3]}} & b;

    // Shift partial products
    assign m1_shifted = m1 << 1;
    assign m2_shifted = m2 << 2;
    assign m3_shifted = m3 << 3;

    // Summation of partial products
    assign s1 = m0 + m1_shifted;
    assign s2 = s1 + m2_shifted;
    assign s3 = s2 + m3_shifted;

    // Final product
    assign p = s3;

endmodule
```

Figure 5.4: multiplier

5.5 4-Bit Divider

```
`timescale 1ns / 1ps
module div4bit(
    input [3:0] a,   // dividend
    input [3:0] b,   // divisor
    output [3:0] q   // quotient
);

    wire [3:0] s0, s1, s2, s3;
    wire c0, c1, c2, c3;
    wire [3:0] b_shift3 = b << 3;
    wire [3:0] b_shift2 = b << 2;
    wire [3:0] b_shift1 = b << 1;

    // Step 1: Compare a >= b << 3
    assign c3 = (a >= b_shift3);
    assign s3 = a - (c3 ? b_shift3 : 0);

    // Step 2: Compare s3 >= b << 2
    assign c2 = (s3 >= b_shift2);
    assign s2 = s3 - (c2 ? b_shift2 : 0);

    // Step 3: Compare s2 >= b << 1
    assign c1 = (s2 >= b_shift1);
    assign s1 = s2 - (c1 ? b_shift1 : 0);

    // Step 4: Compare s1 >= b
    assign c0 = (s1 >= b);
    assign s0 = s1 - (c0 ? b : 0);

    // Final Quotient
    assign q = {c3, c2, c1, c0};

endmodule
```

Figure 5.5: 4-Bit Divison

5.6 Testbench Code

```
`timescale 1ns / 1ps
module tb_arithmetic_operations;
    // Inputs
    reg [3:0] A, B;             // 4-bit operands
    reg [1:0] operation;        // Operation selector
    // Outputs
    wire [7:0] result;          // Output for add, sub, mul, quotient (for div)
    // Instantiate the Arithmetic Operations Module
    arithmetic_operations uut (
        .A(A),
        .B(B),
        .operation(operation),
        .result(result)
    );
    initial begin
        $monitor("Time=%0t | A=%b (%0d), B=%b (%0d), Op=%b | Result=%0d",
                 $time, A, A, B, B, operation, result);

        // Operation Codes:
        // 00 = Addition
        // 01 = Subtraction
        // 10 = Multiplication
        // 11 = Division (Quotient only)

        // Test Case 1: 14 + 1 = 15
        A = 4'b1110; B = 4'b0001; operation = 2'b00;
        #10;
        // Test Case 2: 15 - 8 = 7
        A = 4'b1111; B = 4'b1000; operation = 2'b01;
        #10;
        // Test Case 3: 12 * 14 = 168
        A = 4'b1100; B = 4'b1110; operation = 2'b10;
        #10;
        // Test Case 4: 15 / 5 = 3 (Quotient only)
        A = 4'b1111; B = 4'b0101; operation = 2'b11;
        #10;

        $stop;
    end
endmodule
```

Figure 5.6: testbench code

5.7 Constraint File

```
set_property IOSTANDARD LVCMOS33 [get_ports {result[7]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[6]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[5]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {result[4]}]

set_property PACKAGE_PIN U16 [get_ports {result[0]}]
set_property PACKAGE_PIN E19 [get_ports {result[1]}]
set_property PACKAGE_PIN U19 [get_ports {result[2]}]
set_property PACKAGE_PIN V19 [get_ports {result[3]}]

set_property PACKAGE_PIN W18 [get_ports {result[4]}]
set_property PACKAGE_PIN U15 [get_ports {result[5]}]
set_property PACKAGE_PIN U14 [get_ports {result[6]}]
set_property PACKAGE_PIN V14 [get_ports {result[7]}]


set_property PACKAGE_PIN V17 [get_ports {A[0]}]
set_property PACKAGE_PIN V16 [get_ports {A[1]}]
set_property PACKAGE_PIN W16 [get_ports {A[2]}]
set_property PACKAGE_PIN W17 [get_ports {A[3]}]

set_property IOSTANDARD LVCMOS33 [get_ports {A[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[0]}]
set_property IOSTANDARD LVCMOS33 [get_ports {A[2]}]

set_property IOSTANDARD LVCMOS33 [get_ports {B[3]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[2]}]
set_property IOSTANDARD LVCMOS33 [get_ports {B[0]}]

set_property PACKAGE_PIN W15 [get_ports {B[0]}]
set_property PACKAGE_PIN V15 [get_ports {B[1]}]
set_property PACKAGE_PIN W14 [get_ports {B[2]}]
set_property PACKAGE_PIN W13 [get_ports {B[3]}]

set_property PACKAGE_PIN V2 [get_ports {operation[0]}]
set_property PACKAGE_PIN T3 [get_ports {operation[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {operation[1]}]
set_property IOSTANDARD LVCMOS33 [get_ports {operation[0]}]
```

Figure 5.7: constraint file code

Chapter 6

RTL schematic and Device

6.1 Schematic of FPGA Calculator

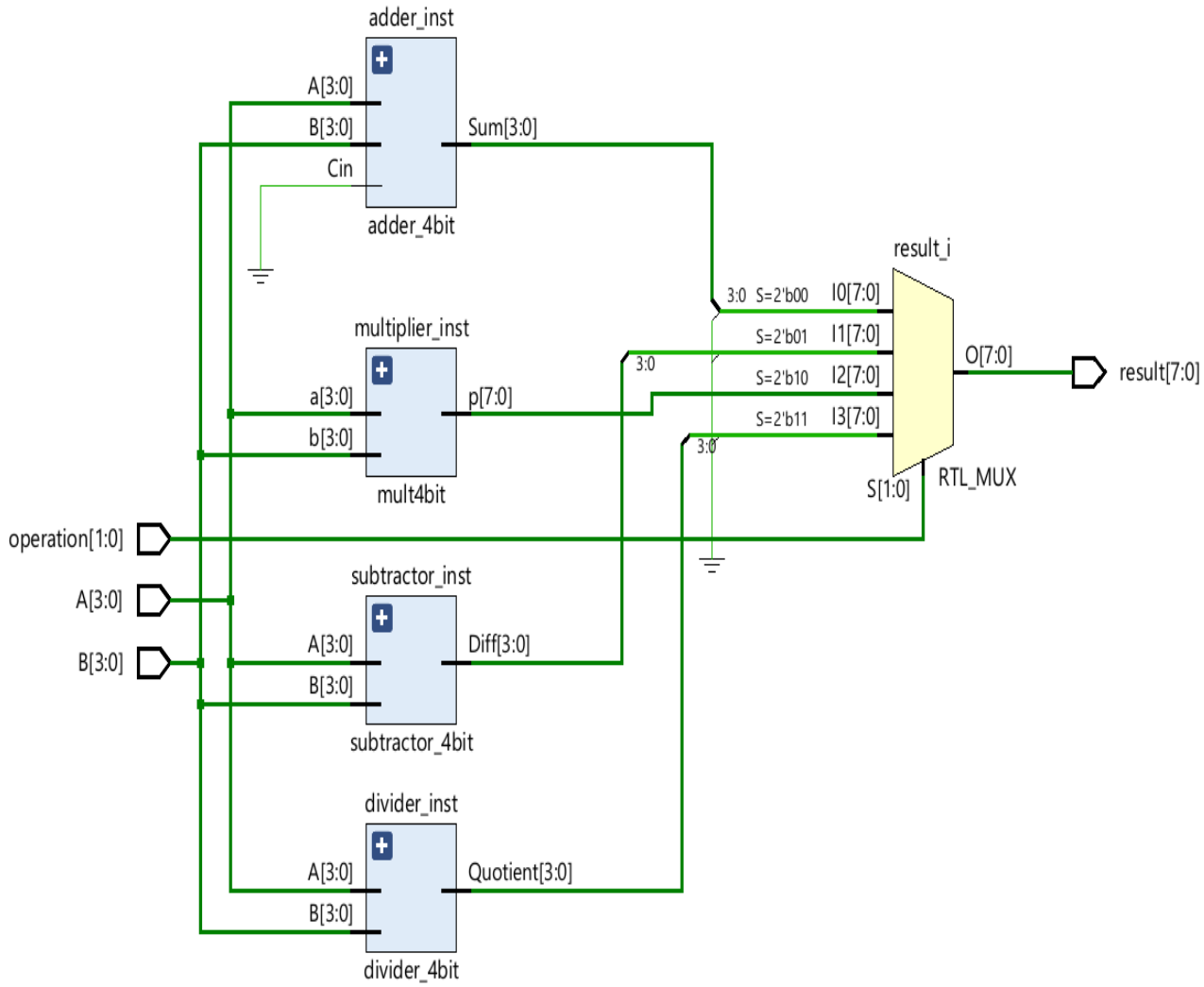

Figure 6.1: schematic view of 4-Bit calculator

6.2 Implemented Design

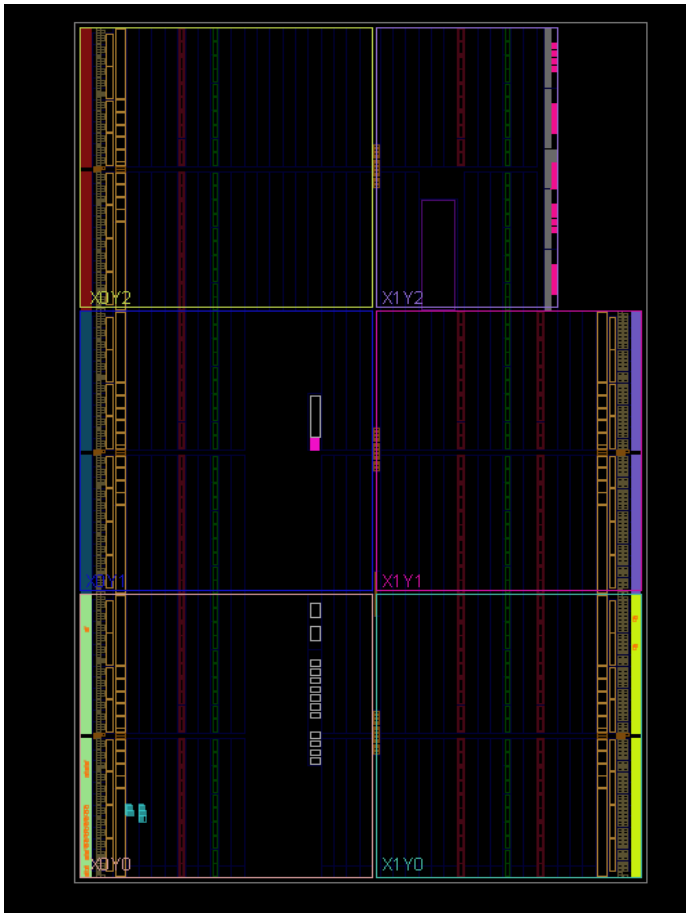

Figure 6.2: Implemented Design

Name	Slice LUTs (20800)	F7 Muxes (16300)	Slice (8150)	LUT as Logic (20800)	Bonded IOB (106)
N arithmetic_operations	39	2	10	39	18
multiplier_inst (mult4bit)	24	1	10	24	0

Figure 6.3: Utilzation Report

Chapter 7

Results and Conclusion

7.1 Testbench Simulation

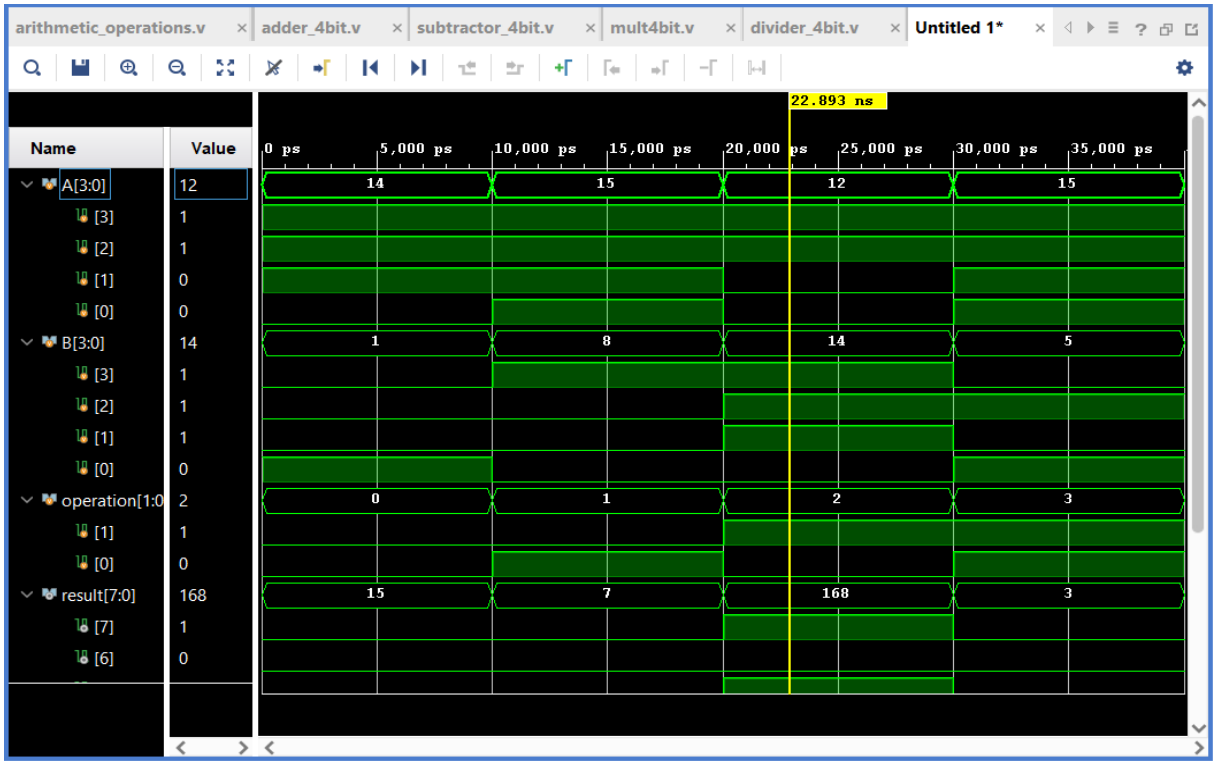

Figure 7.1: Simulation

- 00 – performs addition
- 01 – performs subtraction
- 10 – performs multiplication
- 11 – performs division

7.2 Hardware Results

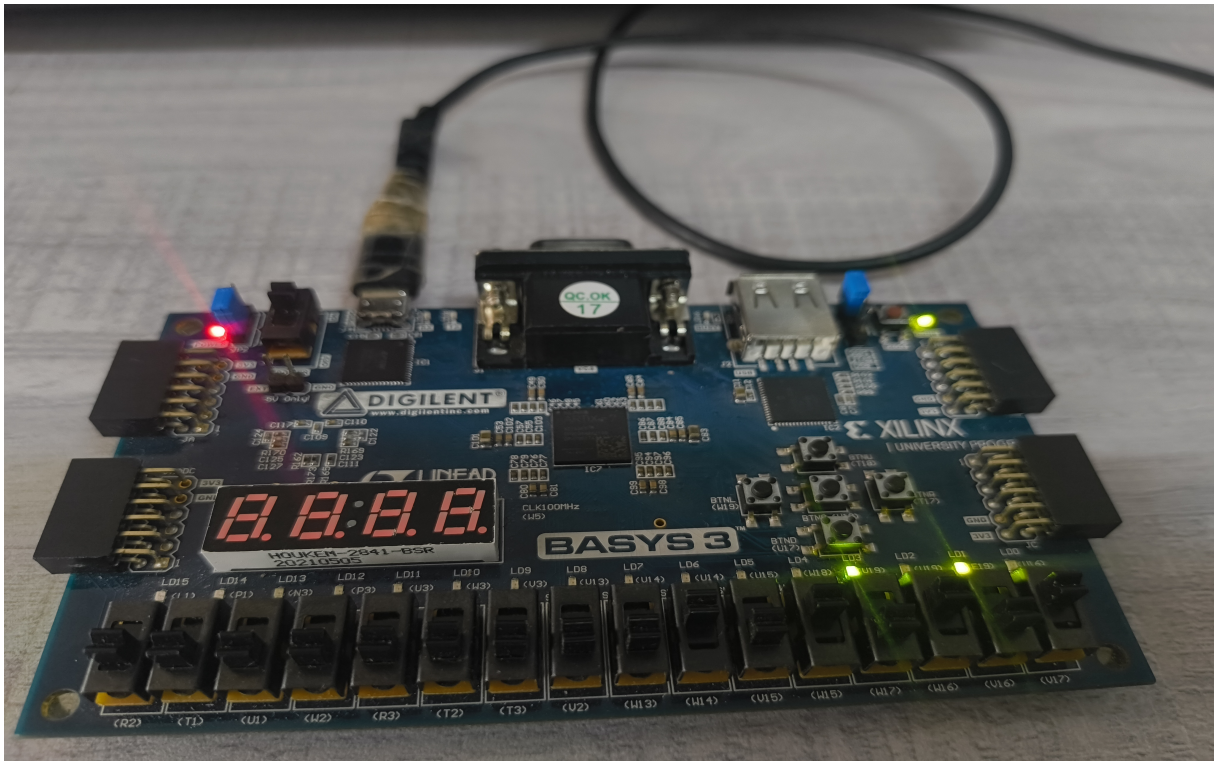

Figure 7.2: 4-Bit Addition

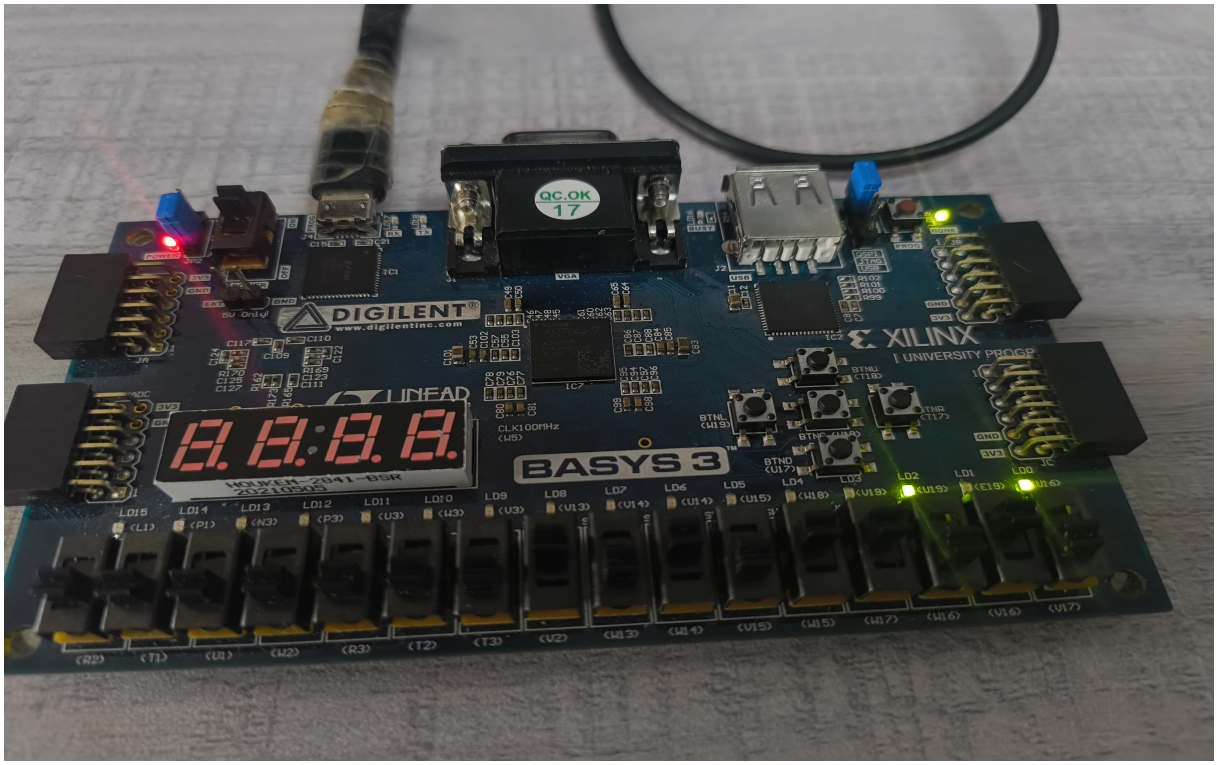

Figure 7.3: 4-Bit Subtraction

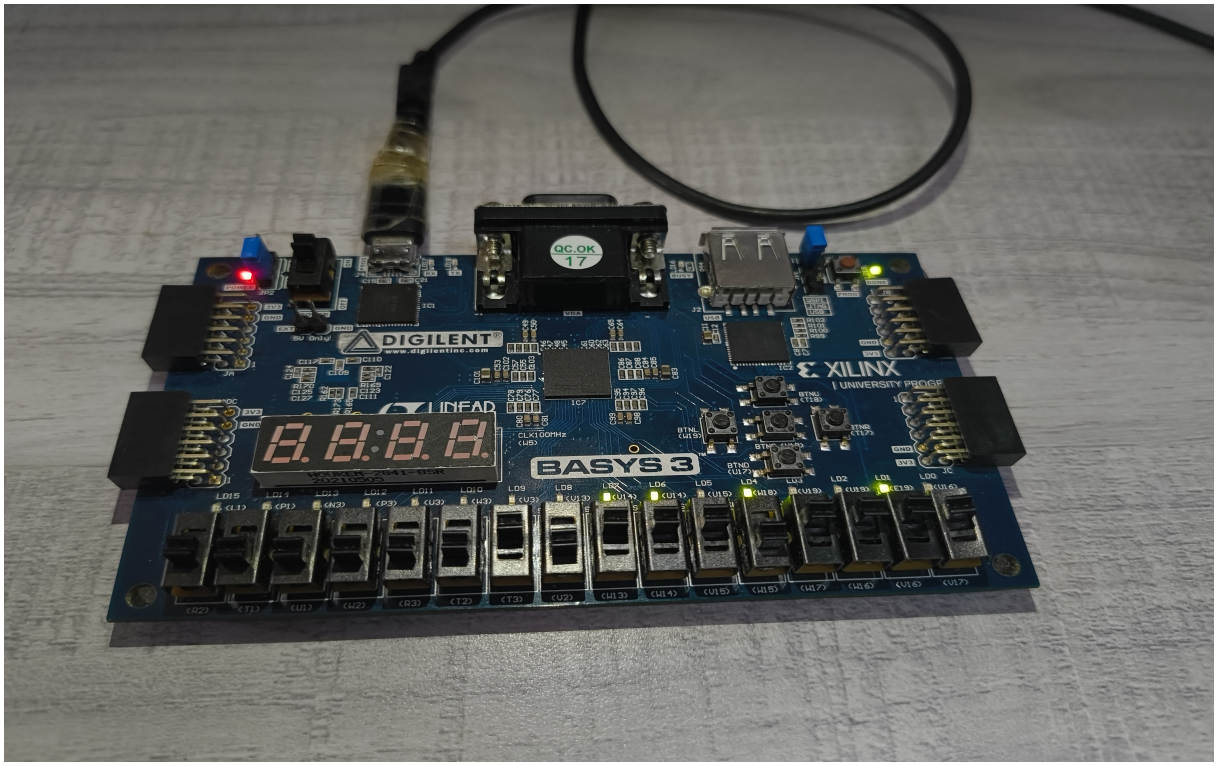

Figure 7.4: 4-Bit Multiplication

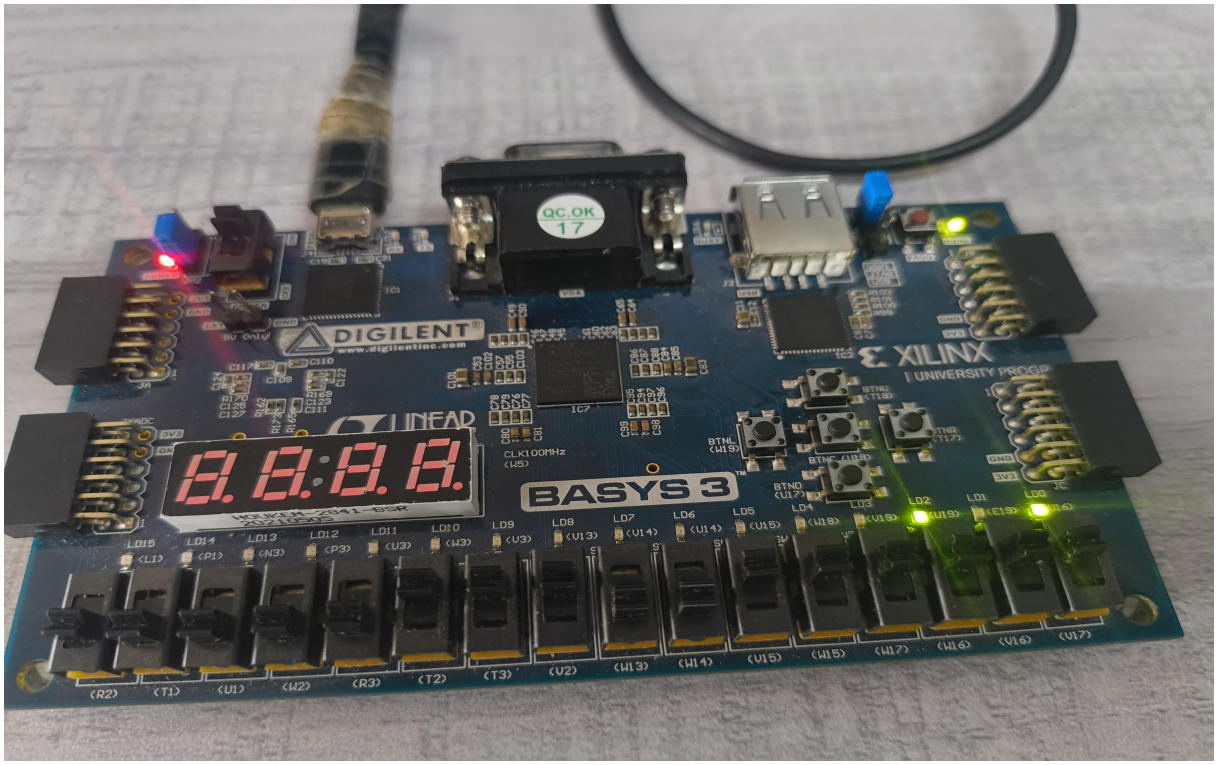

Figure 7.5: 4-Bit Division

7.3 Conclusion

In this project, a 4-bit arithmetic calculator was successfully designed and implemented using the Basys 3 FPGA board and Vivado 2021.2 software. The system is capable of performing four essential arithmetic operations—addition, subtraction, multiplication, and division—based on a 2-bit operation selector. Each arithmetic function was built using fundamental digital design techniques with logic gates. The addition operation was implemented using cascaded 1-bit full adders, while subtraction was realized through full subtractors using basic logic for bitwise borrowing. Multiplication was implemented using the shift-and-add method, generating partial products and summing them using logic gate structures. Division was achieved using combinational logic that computes only the quotient without remainders, following a gate-level approach. The entire system was modeled in Verilog HDL and verified through RTL schematic analysis and behavioral simulation in Vivado, confirming accurate functionality. Input values were provided using the Basys 3 board's switches, and the resulting outputs were displayed directly on the board's LEDs. This project demonstrates a strong grasp of hardware-level arithmetic operation design using gate logic and offers practical insight into implementing a mini arithmetic logic unit (ALU) on FPGA platforms.

Bibliography

[1] S. b. Suhaili, K. J. anak Kumar, N. Julai, M. H. Husin, M. F. M. Sabri and A. Lit, "Implementation of Verilog HDL in Calculator Design with FPGA Simulation," 2020 13th International UNIMAS Engineering Conference (EnCon), Kota Samarahan, Malaysia, 2020, pp. 1-6, doi: 10.1109/EnCon51501.2020.9299337.

[2] A. T. Sarker, G. Datta, L. Mahbub, N. Ahmed and K. M. Daiyan, "A Comparative Analysis of 4-Bit Adder Architectures for Enhanced Efficiency in Conventional CMOS Technology," 2024 IEEE International Conference on Power, Electrical, Electronics and Industrial Applications (PEEIACON), Rajshahi, Bangladesh, 2024, pp. 54-59, doi: 10.1109/PEEIACON63629.2024.10800330.

[3] M. Sanadhya and D. K. Sharma, "Design and implementation of full subtractor using different adiabatic techniques," 2020 IEEE International Women in Engineering (WIE) Conference on Electrical and Computer Engineering (WIECON-ECE), Bhubaneswar, India, 2020, pp. 102-106, doi: 10.1109/WIECON-ECE52138.2020.9397967.

[4] S. Agrawal and D. Patel, "FPGA Implementation and Analysis of Different Multiplication Algorithms," International Journal of Engineering and Advanced Technology (IJEAT), vol. 6, no. 5, pp. 50–53, June 2017.