TRIBHUVAN UNIVERSITY
INSTITUTE OF ENGINEERING
KATHMANDU ENGINEERING COLLEGE

DEPARTMENT OF ELECTRONICS COMMUNICATION AND INFORMATION ENGINEERING

Minor Project Report
On

8-bit ALU Implementation Using FPGA

By:
Bishant Adhikari (075/BEI/007)
Lucky Prajjwal Shrestha (075/BEI/010)
Neetika Aryal (075/BEI/013)

Kathmandu, Nepal
Baisakh 2079

ABSTRACT

For a design to be competitive, its processor has to fit the following characteristics: relatively inexpensive, flexible, adaptable, fast, and reconfigurable. While searching for these vast and complex topics we have forgotten about the ancient mathematical concepts that have laid the foundation for modern mathematical ideas. So we are implementing an 8 bit ALU using ancient Vedic Mathematical Ideas which are surprisingly found to be faster than other Arithmetic Circuits that are implemented today.

A solution to this is the use of Field-Programmable Gate Arrays (FPGA) as design tools. Our project describes the realization of an 8-bit FPGA based Arithmetic and Logic Unit. Our system has been implemented on the FPGA board using ISE foundation 8.1 and VHDL.

ACKNOWLEDGEMENT

We would sincerely like to show our gratitude to our Head of Department of Electronics and Communication **Er. Rajan Lama** Sir for providing us this opportunity to showcase our minor project. We extend our heartfelt gratitude to our project Coordinator **Er. Sujan Shrestha** for coordinating our minor project. We would like to express our special thanks to our Minor Project supervisor **Er. Dhawa Sang Dong** Sir for his support and helping us to tackle different problems faced during this project.

We would also like to send our regards to teachers and individuals who helped us with their support and suggestions for the very idea of this report.

TABLE OF CONTENTS

List of Figures

List of Tables

LIST OF ABBREVIATIONS

ADDR	Address
ALU	Arithmetic Logic Unit
ASCII	American Standard Code for Information interchange
ASIC	Application-Specific Integrated Circuit
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CPU	Central Processing Unit
CRC	Cyclic Redundancy Check
ENDP	End Of Process
EOP	End Of Packet
FPGA	Field Programmable Gate Array
HDL	Hardware Descriptive Language
IC	Integrated Circuits
LCD	Liquid Crystal Display
LUT	Look up table
OTG	On-The-Go
PDA	Personal Digital Assistant
PID	Packet ID
RISC	Reduced Instruction Set Computer
RTL	Register Transfer Level
USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integration Circuit
VLSI	Very Large Scale Integration

1. INTRODUCTION

1.1. Background

The target of our project is to design an 8 bit ALU and implement it on FPGA. Our project describes the realization of an 8-bit FPGA based simple processor i.e. ALU. An arithmetic and logic unit (ALU) is one of the most important components in a microprocessor, and is typically the part of the processor that is designed first. Once the ALU is designed, the rest of the microprocessor is implemented to feed operands and control codes to the ALU.

All the parts of an ALU are designed using VHDL and then structured to get the top design of the existing ALU. Digital circuits captured using VHDL can be easily simulated, are more likely to be synthesizable into multiple target technologies and can be archived for later modification and reuse. Due to this reason we chose the VHDL language.

The enhancements in hardware technology have led to the existence of Field Programmable Gate Array (FPGA) that is large enough to accommodate a complete system on a single device. The single chip-design allows designers to place a large number of functions onto a system on a programmable chip and to reprogram this chip from the desktop thus removing engineering costs from prototyping and testing of new designs. The use of hardware description languages (HDLs) allows FPGAs to be more suitable for different types of designs where errors and component failures can be limited.

1.2. Problem statement

In a world of rapidly innovating technology, rural parts of our country still don't have access to digital electronics and expensive equipment that comes along with it. With the help of FPGA we can eliminate maximum of those problems with minimum of costs. We can demonstrate implementation of most traditional and basic Digital Circuit Boards in an ALU using a single FPGA Development Board. Specific projects may require specific circuits which cannot be implemented optimally using commercially available boards and thus FPGA comes into play.

1.3. Objectives

- To design and verify different operations of Arithmetic and Logic Unit.
- To understand the basics of FPGA and VLSI design.

1.4. Scope of Project

Based on available resources, limited time frame and expertise, this research project is narrowed down to the following scopes of the work.

- This project includes the design of 8-bit ALU and its components using VHDL and collecting the simulation result.
- Analysis of the simulated result of 8-bit ALU and implement using the FPGA.
- This project demonstrates a few simple instructions to execute.

1.5. Applications

- Can perform very fast mathematical operations parallelly for an application specific design, which otherwise conventional CPU designs can't handle.
- We can use this ALU Design as a building block in various other projects .

2. LITERATURE REVIEW

International Journal of Advanced Research in Computer Science and Software Engineering published the report on 'Control Unit Design of a 16-bit Processor Using VHDL' by Alpesh Dauda, Nalinikanta Barpanda, Nilamani Bhoi and Manoranjan Pradhan, describes the design and implementation of control unit of a 16-bit processor that is implemented on the FPGA device. Design entry of the control unit is done by using VHDL code. From the synthesis estimate,the minimum clock period that can be achieved in this control unit architecture is 6.646 ns,which translates to the maximum operating frequency of 150.466 MHz and the combinational path delay is 8.549 ns.[1]

A Report on 'Design of 4-bit ALU Using Logic Gates' prepared by Mahesh Neupane, Prem Pun and Amar NathDeo, the students of the Nepal Engineering College covers all the basic components of the ALU. They use different logic gates to design a circuit. In this project, they design an ALU with eight arithmetic operations and four logic operations. They implement the combinational logic in their project. For the output of the operations they use seven segment displays. The outputs of their circuit are usually in 8 bit BCD form, and are not suitable for directly driving seven segment displays. The special BCD to seven segment decoder ICs is used to convert the BCD signal into a form suitable for driving these displays.[2]

Amity School Of Engineering and Technology published the report on 'Designing a 8 bit ALU and Implementing On Xilinx Vertex 4 FPGA' by Preeti Takhar, Priyanka Rajpal, Rahul Borthakur and Sakshi Agarwal, describes the design and implementation of 8 bit ALU that is implemented on the Xilinx Vertex 4 FPGA device. They designed an 8-bit ALU which accepts two 8 bit numbers and the code corresponding to the operation which it has to perform from the user. To implement the ALU the coding was written in the VHDL and verified in ModelSim. They obtained the waveforms successfully. The synthesis of the code was performed using Xilinx-ISE. Thereafter, the simulation was done to verify the synthesized code. And it was converted into binary format. Finally, they successfully verified the code using FPGA.[3]

However, in our project we have designed an 8-bit ALU and Matrix Multiplier using an FPGA board, written in VHDL. Moreover, what makes our project different from the rest is that we are applying Vedic algorithms for multiplication. Though it seems to be just another religious book, it contains tons of mathematical ideas. Out of which we are using "Urdhva Tiryakbhyam Sutra" for multiplication. This formula makes our project more appealing, interesting and faster.

3. THEORETICAL BACKGROUND

3.1. Introduction to VHDL

VHDL is a hardware description language. It is a language specifically developed to describe digital electronic hardware and its attributes. It is a flexible language and can be applied to many different design situations. It is industry standard language for electronic design automations used to describe both digital and mixed-signal systems such as FPGA and Integrated Circuits.

3.1.1. Why use VHDL?

Although VHDL is considered tougher to understand then other hardware description languages, we have chosen to implement our project in VHDL as it offers the following advantages:

- VHDL is an IEEE and ANSI standard. This increases portability between tools, components and products. It means that VHDL hardware designs and testbenches are portable across the multiple platforms of various design tools available in the market.
- VHDL was designed to be technology independent implying that if a system was described today using VHDL, then it can be implemented using any previous or future technology if available.
- Behavioral simulations can reduce design times by allowing the design problems to be detected early on, thereby avoiding the need to re-work designs at the gate level.
- It supports both synchronous and asynchronous timing models.

3.2. Introduction to FPGA

Field Programmable Gate Arrays (FPGAs) are semiconductor devices that are based around a matrix of configurable logic blocks (CLBs) connected via programmable interconnects and surrounded by an array of I/O blocks. It can be designed by Hardware Description languages like verilog and VHDL. It can be reprogrammed until the processor design is final.

We used the Xilinx Spartan3E-500 FG320 FPGA in the Nexys-2 board from Digilent in our project. Each CLB in the Spartan-3E FPGA contains four slices, each of which contains two 16 x 1 RAM look-up tables (LUTs), which can implement any combinational logic function of

four variables. In addition to two look-up tables, each slice contains two D flip-flops which act as storage devices for bits. This chip contains 1,164 CLBs arranged as 46 rows and 34 columns. There are therefore 4,656 slices with a total of 9,312 LUTs and flip-flops. This part also contains 368,640 bits of block RAM. Half of the LUTs on the chip can be used for a maximum of 74,752 bits of distributed RAM. The basic architecture of a Spartan-3E FPGA is shown in fig. below:

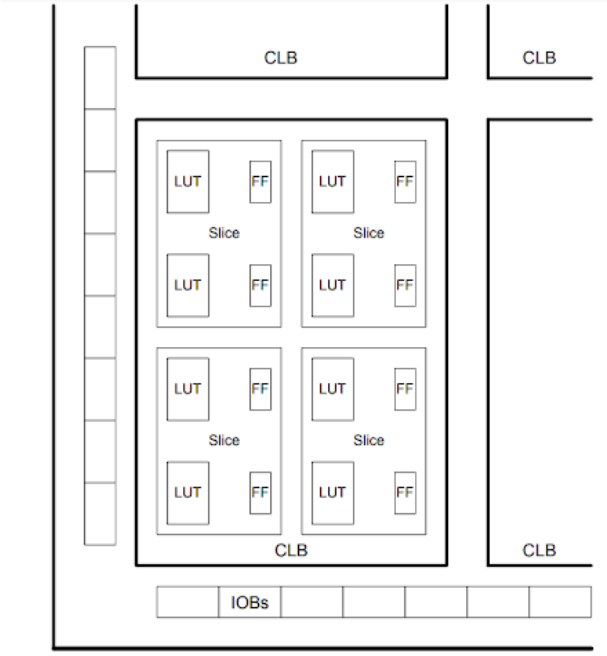

Figure 3.1: Architecture of Spartan-3E FPGA

3.2.1. Why use FPGA?

- Very fast on chip demonstration
- Simple and fast design process
- High performance
- Re-programmable

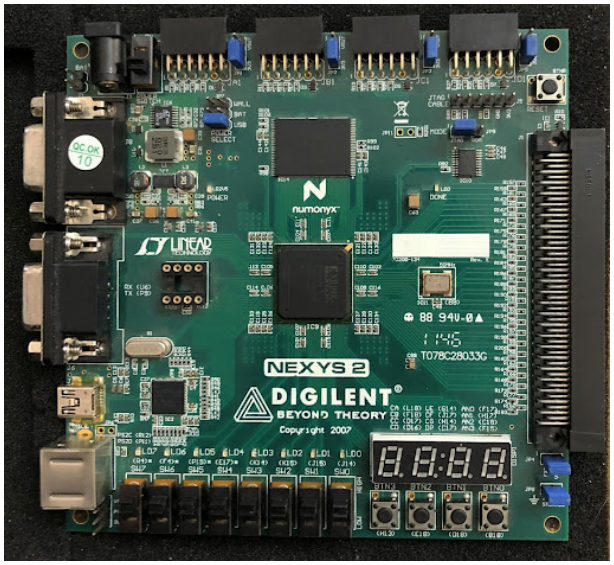

Figure 3.2: Nexys-2 board from Digilent with a Xilinx Spartan 3E-500 FG320 FPGA

3.3. Major steps in FPGA programming

The major steps in FPGA programming are:

1. Hardware architecture design.

   FPGA has a SoC architecture. ”System On a Chip” is an integrated circuit that contains all the required circuitry and components of an electronic system on a single chip.

2. Design

   This is the process of creating the hardware logic itself, typically by writing register-transfer logic (RTL) using a hardware description language such as VHDL or Verilog. The goal is to match the functionality of the algorithm while operating on a continuous stream of data, using fixed-point operations for efficiency.

3. Verification

   This step ensures that the design works as intended before FPGA programming. This can be as simple as a VHDL or Verilog. Commercial projects typically use a methodology such as the Universal Verification Methodology (UVM).

4. Synthesis

   This technology transforms the RTL to digital logic gates and attempts to meet your register-to-register clock frequency goals while minimizing use of the resources on the FPGA.

5. Integration

   An FPGA contains a lot of dedicated resources already — the pins, the clock signal, input/output processing such as analog-to-digital converters (ADC), and interfaces to off-chip memory and other devices on the board. An FPGA also has dedicated registers that both the hardware and software can use to communicate with each other. Your design will need to plug into this "reference design."

6. Implementation

   This is the process of determining which physical resources on the FPGA to program with which logic, and how to connect (route) them. This produces the bitstream that is loaded onto the device for FPGA programming.

7. Testing and Debugging

   After FPGA programming, you can run using real input or test input. The first few tries often involve figuring out why it does not work and how to fix it. Most of the time this is due to problems in the design step that were not identified in the verification step.

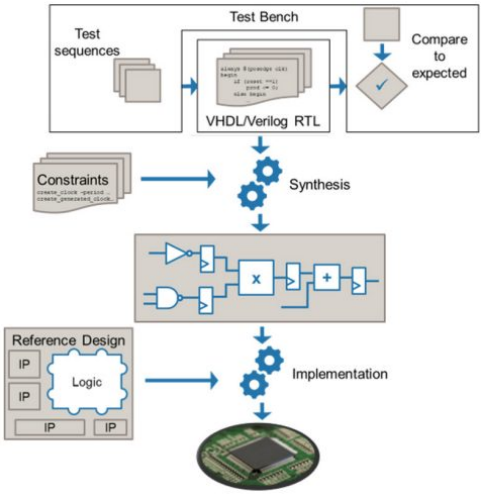

Figure 3.3: A typical workflow from algorithm design to FPGA programming

4. METHODOLOGY

4.1. Block Diagram of the System

The figure below shows the block diagram of a simple 8 bit Arithmetic And Logic Unit along with input and output media.

The ALU itself is also a combination of combinational circuits and Finite State Machines designed in VHDL. It consists of basic Arithmetic Operations like Addition Multiplication Division and Logical Operations like AND NOR etc. The ALU gets the input from registers A and B and instruction as a signal from the control unit. It performs the operation and stores the output in the output register Y and updates the Flag Register.

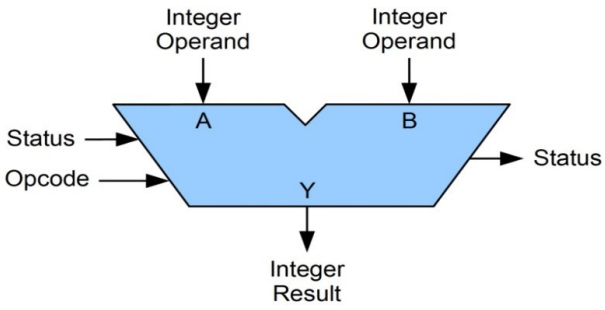

Figure 4.1: ALU block with 2 inputs and an output

4.2. Instruction Design

All instructions are 5 bit instructions for the ALU. However the input for the system is in English and mathematical expression the decoders before the ALU are tasked with the necessary conversion of mnemonics to opcode and data bytes.

When the control unit decodes the input from the keyboard it generates opcode for the instruction along with the data bytes to be operated on and assigns the data bytes to register A and B along with the generated opcode which is used to generate required control signals to activate the circuit of choice.

The instructions to be implemented and their opcodes are listed in the table given below. To reduce the complexity, we want to design the processor which is not pipe-lined. Some of the possible sets of instructions that can be implemented in our proposed processor are listed below.

Opcode	Mneumonics	RTL Description	Functional Description
0110	F, A, B	F←A + B	Add A and B storing the result in F
0111	F,A,B	F←A - B	Subtract B from A storing the result in F
1000	F,A,B	F←A * B	Multiply A and B storing the result in F
0000	F,A,B	F← A AND B	Bitwise anding of A and B storing the result in F
0001	F,A,B	F← A OR B	Bitwise oring of A and B storing the result in F
0011	F,A,B	F← A NOR B	Bitwise nor of A and B storing the result in F
0101	F,A,B	F← A XOR B	XOR the contents of A and B , storing the result in F
0010	F,A,B	F← A NAND B	Bitwise nand of A and B storing the result in F

Table 4.1: Instruction Architecture

4.3. Overview of the Design

4.3.1. Arithmetic and Logic Unit (ALU)

The ALU is the fundamental building block of the central processing unit of a computer. Depending on how the ALU is designed it can make the CPU more powerful. It performs the number of arithmetic operations such as add, subtract, multiply and some logic operations such as AND, OR, XOR and so on. In the block diagram of the ALU there are two input lines a and b which are 16 bit long. The opcode line selects which operation is to be performed from the several operations. When the carry is generated the carry flag is set, and when there is an overflow during the operations the overflow flag is set. The test flag is used to compare the two values. ALU includes several blocks such as adder, logical, tester etc. this block is shown in the RTL simulation of the ALU part.

4.3.1.1 Addition and Substraction

the additionand substraction circuit is implemented using ripple carry adder.The 8-bit ripple carry adder consists of eight full adder cells in cascade such that output carry of one full adder cell is applied as an input carry to another full adder cell The Architecture of an 8-bit ripple carry adder is shown in the above figure. Eight inputs a7 to a0 and b7 to b0 are applied to each of the full adder cells and output S7 to S0 represents eight bit sum from each full adder. The input carry of the first half adder cell must be grounded for the correct addition of least significant bits (a0,b0) otherwise it will result in erroneous output. As the whole computation solely relies on carry rippling, it results in a large value of delay overhead. However, ripple

carry adder is one of the simplest adder architectures and requires less power consumption and area.

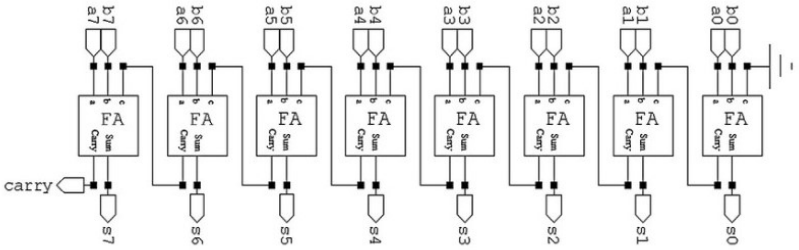

Figure 4.2: Ripple carry adder

For Simplicity we have used an array of complementors to implement Addition and Subtraction in the same circuit.

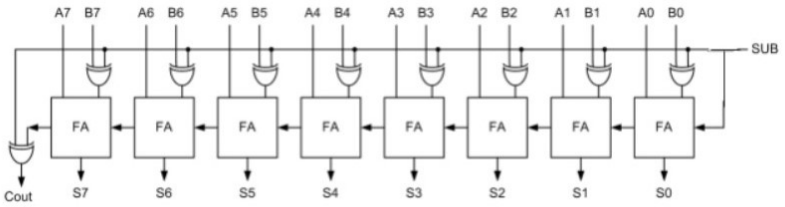

Fig 4.3: Subtractor

Figure 4.3: Substractor

If the value of SUB (Control line) is 1, the output of B0 (xor) SUB = B0(Complement B0). Thus the operation would be A+(B0). Now 2's complement subtraction for two numbers A and B is given by A+B'. This suggests that when SUB=1, the operation being performed on the four bit numbers is subtraction.

4.3.1.2 Multiplication .

Urdhva Tiryagbhyam Sutra

As the multiplication operation in UT sutra is computed parallel, the delay in output is significantly reduced. The sutra is also known as 'vertically and crosswise'. Initially the UT sutra was used for decimal numbers only. However it can also be used for binary numbers.

2x2 bit Vedic Multiplier

Consider the following two, 2 bit numbers A and B where A = a1a0 and B = b1b0 as shown in Fig. 3. Firstly, the least significant bits a0 and b0 are multiplied which gives the least significant

bit s0 of the final product (vertical). Then, the LSB of the multiplicand is multiplied with the next higher bit of the multiplier and added with, the product of the LSB of the multiplier and next higher bit of the multiplicand (crosswise). The sum gives the second bit s1 of the final product and the carry is added with the partial product obtained by multiplying the most significant bits to give the sum s2 and carry c2. The sum is the third corresponding bit and carry becomes the fourth bit of the final product.

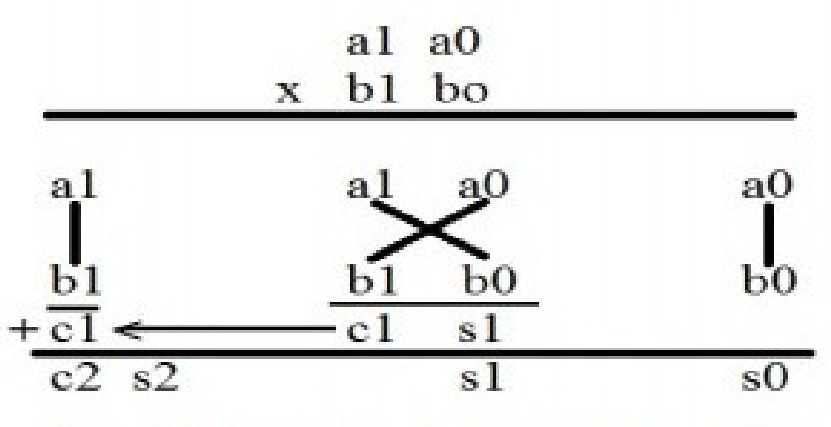

Figure 4.4: Multiplication of two 2 bit binary numbers

$$s0 = a0b0 \tag{1}$$

$$c1s1 = a1b0 + a0b1 \tag{2}$$

$$c2s2 = c1 + a1b1 \tag{3}$$

So the final output is c2 s2 s1 s0. To add the sums in eq: 2 and eq 3 we use two half adders.

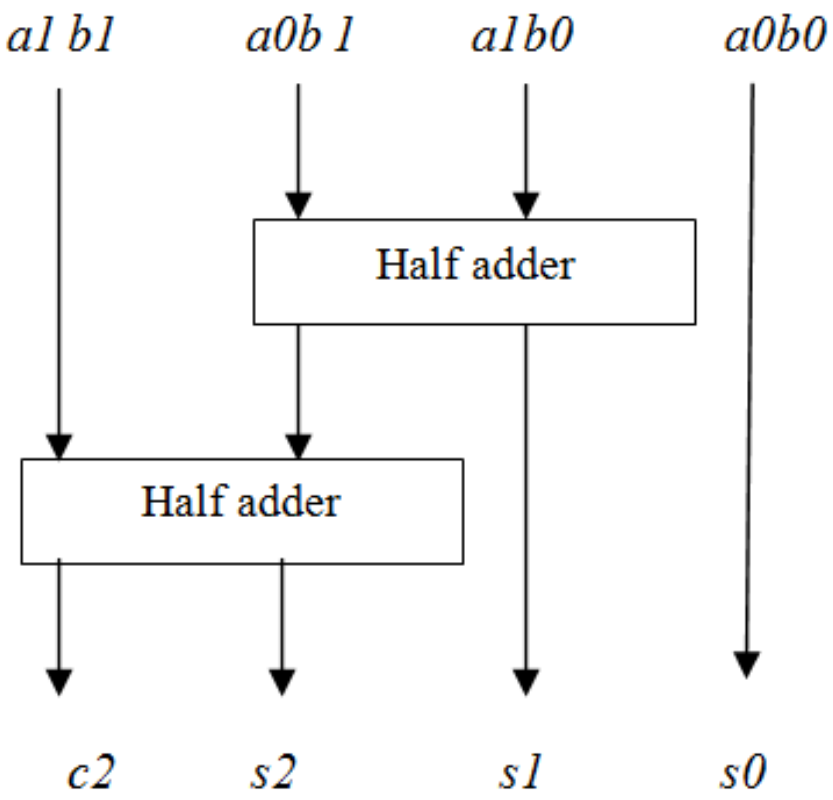

Figure 4.5: 2x2 Vedic Multiplier Unit

4x4 bit Vedic Multiplier

The 4x4 bit Vedic multiplier is implemented using four 2x2 bit Vedic multiplier blocks as discussed in Fig. 6. Let's assume the two 4 bit numbers, say A= A3 A2 A1 A0 and B= B3 B2 B1 B0. The output line for the multiplication result is – S7 S6 S5 S4 S3 S2 S1 S0. Divide A and B into two parts, say A3 A2 and A1 A0 for A and B3 B2 and B1 B0 for B. Now assume two bits together and consider it a single bit then, the number of bits to be multiplied will be now 2x2. The multiplication results will be 4 bits which are composed of two bits together and their separate multiplication is to be done using a 2x2 multiplier.

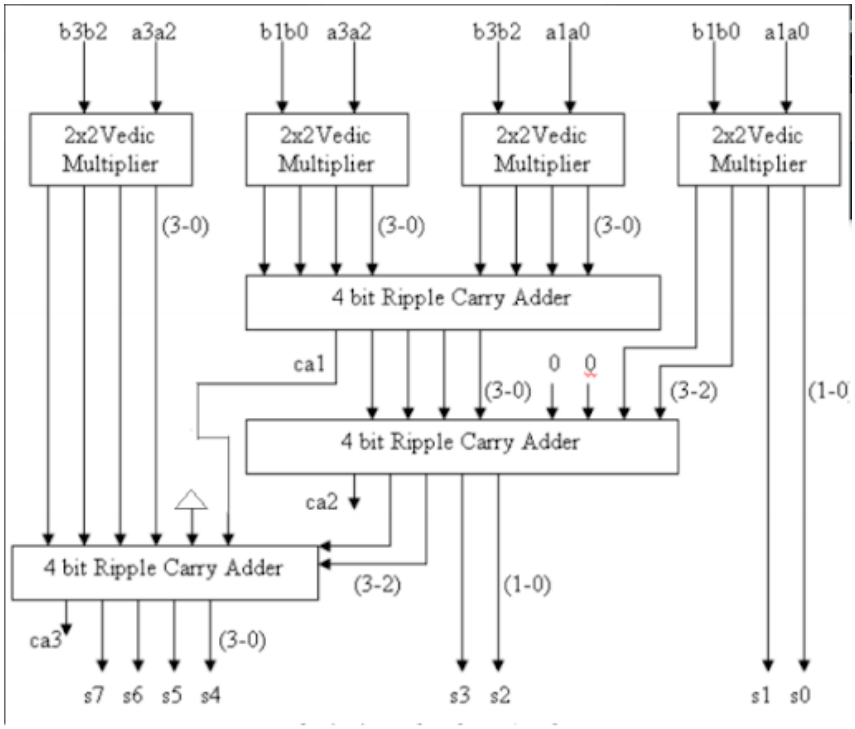

Figure 4.6: 4x4 Vedic Multiplier Unit

8x8 bit Vedic Multiplier

The 8x8 bit Vedic multiplier can be easily implemented by using four 4x4 bit Vedic multiplier blocks as shown in the block diagram in Fig. 7 which was discussed in the previous section. Let's assume two numbers for 8x8 multiplications, say A= A7 A6 A5 A4 A3 A2 A1 A0 and B= B7 B6 B5 B4 B3 B2 B1B0. In the multiplication result the output bits will be of 16 bits as – S15 S14 S13 S12 S11 S10 S9 S8 S7 S6 S5 S4 S3 S2 S1 S0. Now assume A and B into two parts, say the 8 bit multiplicand A is made of 4 bits AH-AL each. Similarly multiplicand B is made of two parts of 4 bits BH-BL respectively. The 16 bit result can be given as: P = A x B = (AH-AL) x (BH-BL) = AH x BH + (AH x BL + AL x BH) + AL x BL Using the fundamental of Vedic multiplication, taking four bits at a time and using 4 bit multiplier block as discussed we can perform the multiplication. The outputs of 4x4 bit multipliers are added accordingly to obtain the final product. Here total three 8 bit Ripple-Carry Adders are required as shown in Fig 7.

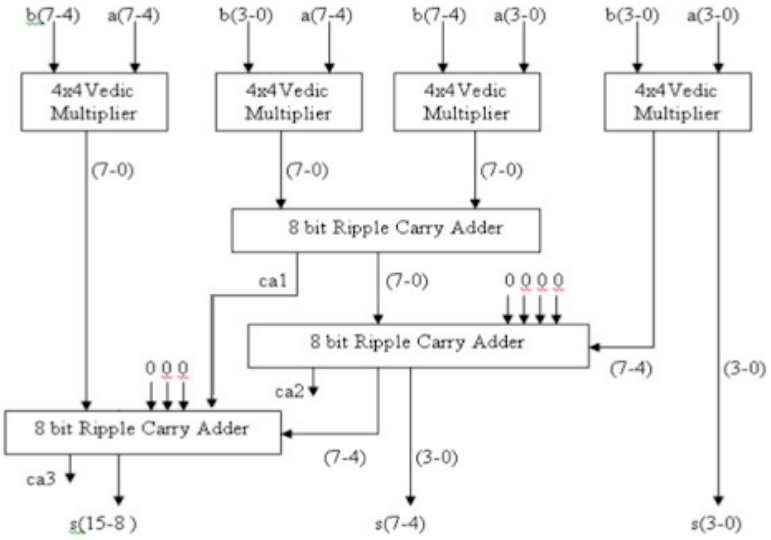

Figure 4.7: 8x8 Vedic Multiplier Unit

4.3.1.3 Logical Block

The logical operations such as AND, XOR, NOT operations etc. are handled by the logical block of the ALU. It takes two 8 bit data register inputs and multiple control signals for each operation to be performed and outputs to the 16 bit output register which is a generic register for all operations. Only the lower 8 bit of the output register is utilized for logical operations.

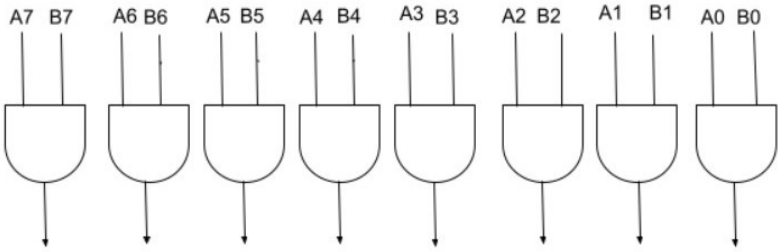

Figure 4.8: 8-Bit AND Block

The figure above shows a 8 bit AND Block made by simply adding 8 traditional AND gates which take parallel input and produce parallel output. Similarly all the other gates will be implemented by adding 8 of the traditional 1 bit logical gates for each logical operation.

4.3.2. Matrix Multiplication

In mathematics, particularly in linear algebra, matrix multiplication is a binary operation that produces a matrix from two matrices. For matrix multiplication, the number of columns in the first matrix must be equal to the number of rows in the second matrix. The resulting matrix, known as the matrix product, has the number of rows of the first and the number of columns of the second matrix. The product of matrices A and B is denoted as AB.

4.3.2.1 3x3 8-bit Matrix Multiplication using Vedic Multiplier

The Multiplication algorithm used can be used to make a matrix multiplier which would be its primary use as matrix multiplication is very very important in fields like Computer Graphics and Machine Learning.

For the inputs and outputs of the matrix multiplier we will be using a stream of bits produced by concatenating the individual 8 bit values sequentially.

Let us take an example.

$$X = Matrix1 * Matrix2$$

$$X = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \cdot \begin{bmatrix} 5 & 76 & 7 \\ 12 & 156 & 65 \\ 35 & 34 & 65 \end{bmatrix}$$

$$X = \begin{bmatrix} 134 & 490 & 332 \\ 878 & 8932 & 3928 \\ 866 & 7546 & 3429 \end{bmatrix}$$

Here the two input matrices have 9 elements 8 bit wide each so total bits are 72 bits and the output is 16x9 which is 144 bits.

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 12 \\ 35 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 76 \\ 156 \\ 34 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 7 \\ 65 \\ 65 \end{bmatrix}$$

And so on. The Individual dot products then can be carried out as

$$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix} \cdot \begin{bmatrix} 5 \\ 12 \\ 35 \end{bmatrix} = 1 \cdot 5 + 2 \cdot 12 + 335 = 134$$

We can see that one dot product uses three 8 bit multipliers and two 16 bit adders to get the output. We instantiate 9 different dot product blocks parallelly containing 3 multipliers and 2 adders each to get a matrix multiplier.

For Inputs and Output we will have a total of 3x8 which is 24 bits for each input and a 16 bit output to store the dot product.

4.3.2.2 Dynamic creation of mxn Matrix Multiplication

Different applications of matrix multiplication require custom matrix sizes, so we are using a Python script to synthesize the VHDL code. The python script takes in the dimension of the matrices to be multiplied and generates the dot product and matmul code which can be used alongside the code from the ALU to generate the bit file to program the FPGA.

5. SIMULATION WORKS

We have simulated the VHDL code of the 8 bit ALU using the software tool as described below. The 8 bit ALU is coded with VHDL. The Input and Output Devices is not simulated however the necessary input and output is visualized as timing diagrams, RTL Diagrams produced from the ALU Simulation independently.

5.1. Xilinx ISE design suite

Vivado Design Suite is a software suite produced by Xilinx for synthesis and analysis of 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).

Vivado includes the in-built logic simulator ISIM Vivado also introduces high-level synthesis, with a toolchain that converts C code into programmable logic.We used the ISE design suite to compile and run our VHDL code after many rounds of simulation, testing, and analysis on a Xilinx Spartan 3E-500 FG320 FPGA in the Nexys-2 development board.

5.2. ADEPT Digilent

Digilent Adept is a unique and powerful solution that allows you to communicate with Digilent system boards and a wide assortment of logic devices.

- Configure the Xilinx logic devices. Initialize a scan chain, program FPGAs, CPLDs, and PROMs, organize and keep track of your configuration files
- Transfer data to and from the onboard FPGA on your system board. Read from and write to specified registers. Load a stream of data to a register or read a stream of data from a register.
- Organize and quickly connect to your communications modules.
- Program Xilinx XCFS Platform Flash devices using .bit or .mcs files.
- Program Xilinx CoolRunner2 CPLDs using .jed files.
- Program most Spartan and Virtex series FPGAs with .bit files.

6. DESIGN AND ANALYSIS

6.1. AND Gate

A bitwise AND takes two binary representations of equal length and performs the logical AND operation on each pair of corresponding bits. In each pair, the result is 1 if the first bit is 1 AND the second bit is 1. Otherwise, the result is 0.

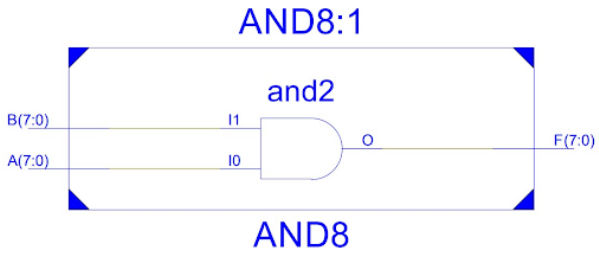

Figure 6.1: RTL diagram of AND gate

6.2. OR Gate

A bitwise OR takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits (the first of each; the second of each; and so on) and performing the logical inclusive or operation on each pair of corresponding bits. In each pair, the result is 1 if the first bit is 1 OR the second bit is 1 OR both bits are 1, and otherwise the result is 0.

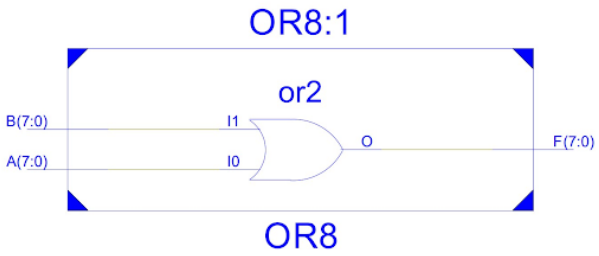

Figure 6.2: RTL diagram of OR gate

6.3. XOR Gate

A bitwise exclusive or takes two bit patterns of equal length and performs the logical XOR operation on each pair of corresponding bits. The result in each position is 1 if the two bits are

different, and if they are the same.

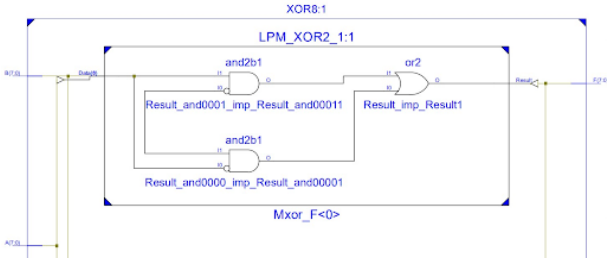

Figure 6.3: RTL diagram of XOR gate

6.4. NAND Gate

A bitwise NAND takes two binary representations of equal length and performs the logical NAND operation on each pair of corresponding bits. In each pair, the result is 0 if the first bit is 1 AND the second bit is 1. Otherwise, the result is 1.

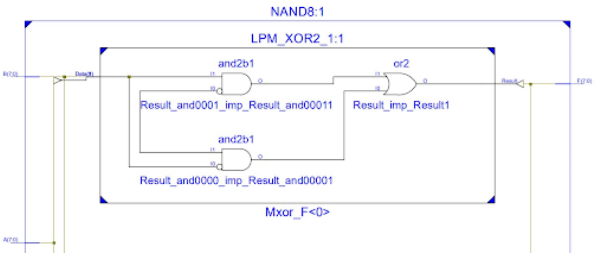

Figure 6.4: RTL diagram of NAND gate

6.5. NOR Gate

A bitwise NOR takes two bit patterns of equal length, and produces another one of the same length by matching up corresponding bits (the first of each; the second of each; and so on) and performing the logical inclusive OR operation on each pair of corresponding bits. In each pair, the result is 1 if the both bits are zero otherwise the result is 0.

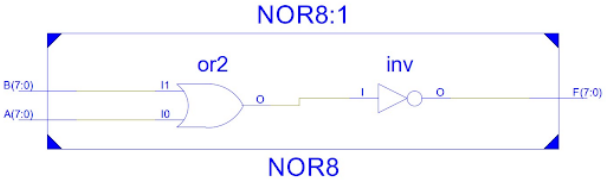

Figure 6.5: RTL diagram of NOR gate

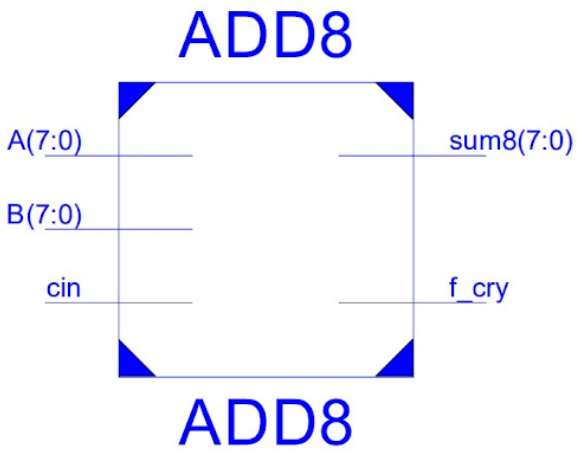

Figure 6.6: Black box diagram of addition

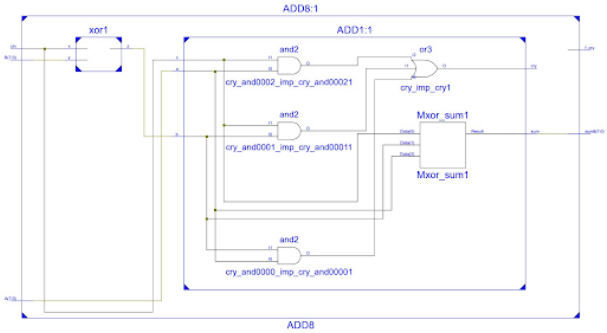

Figure 6.7: RTL diagram of addition

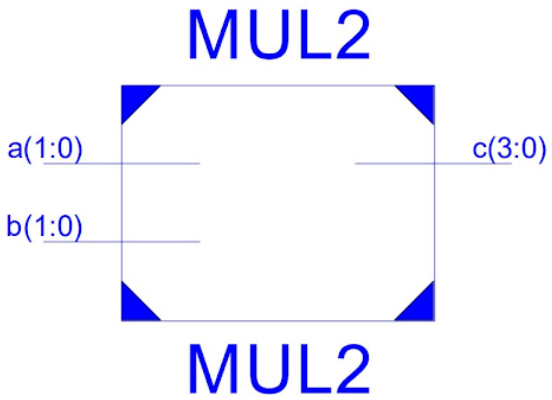

Figure 6.8: Black box diagram of 2-bit Multiplier

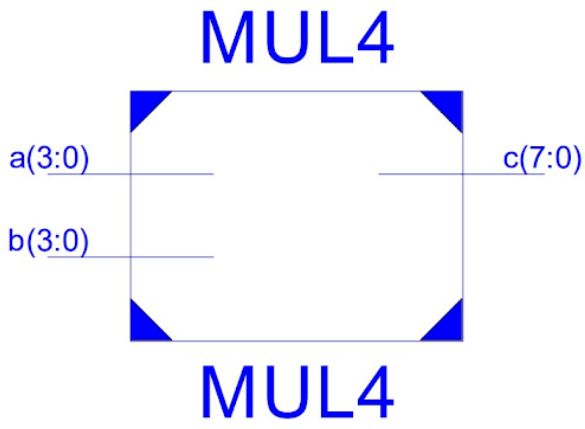

Figure 6.9: Black box diagram of 4-bit Multiplier

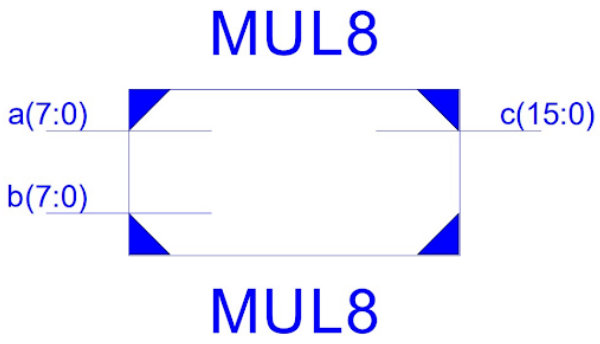

Figure 6.10: Black box diagram of 8-bit Multiplier

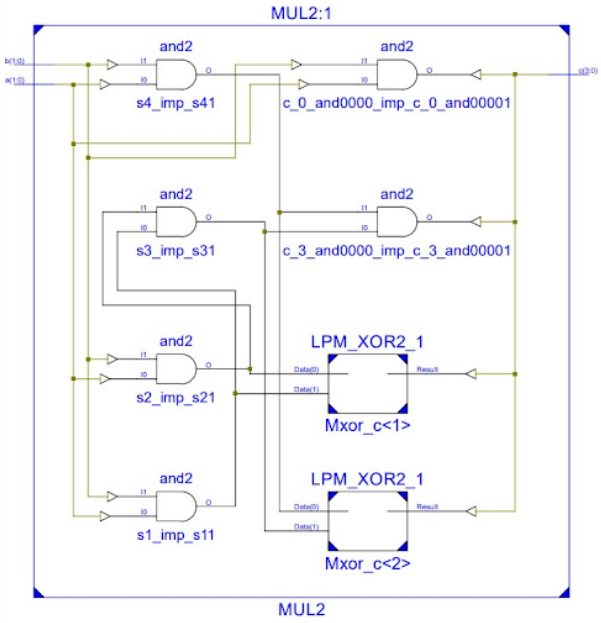

Figure 6.11: RTL diagram of 2-bit Multiplier

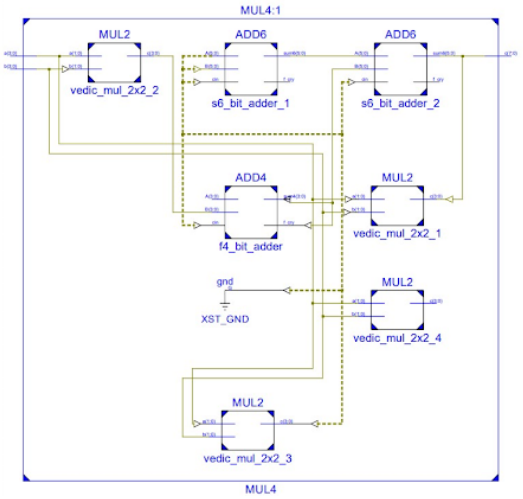

Figure 6.12: RTL diagram of 4-bit Multiplier

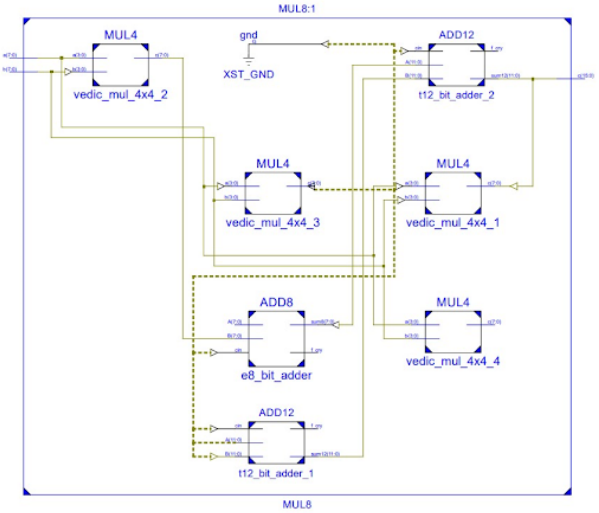

Figure 6.13: RTL diagram of 8-bit Multiplier

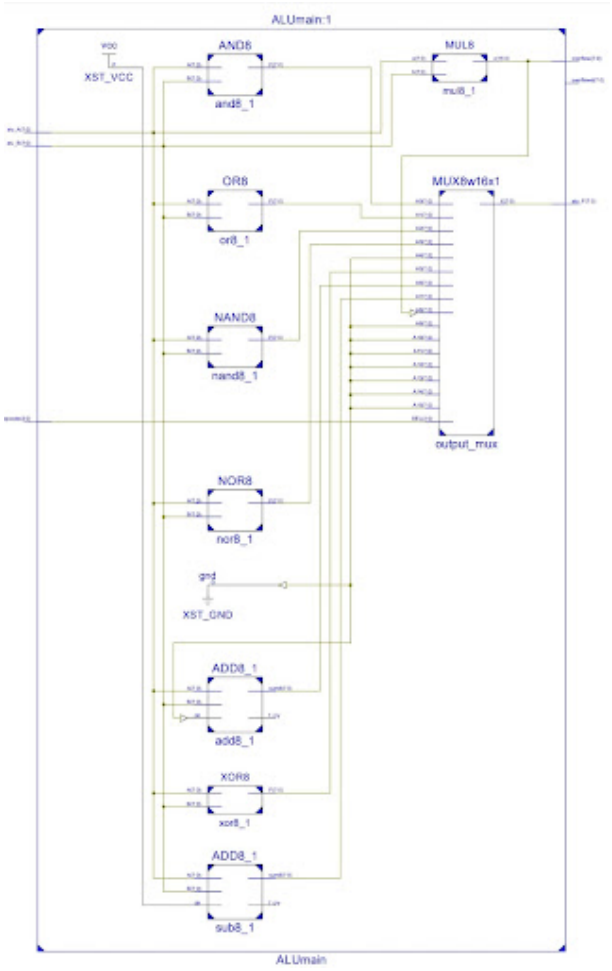

Figure 6.14: ALU Block Diagram

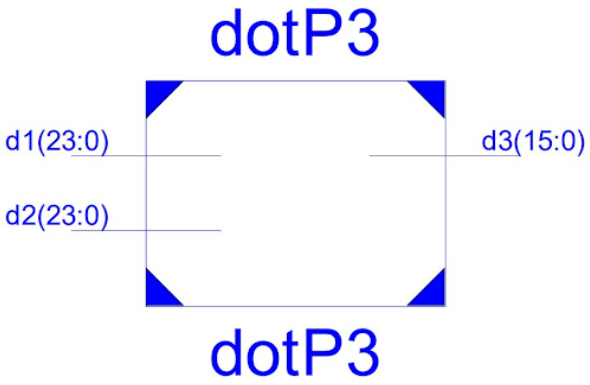

Figure 6.15: Black Box Diagram of Dot product module

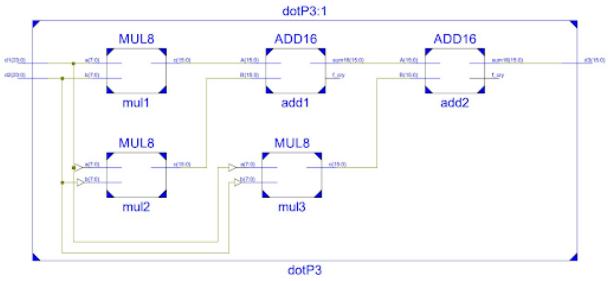

Figure 6.16: RTL Diagram of Dot product module

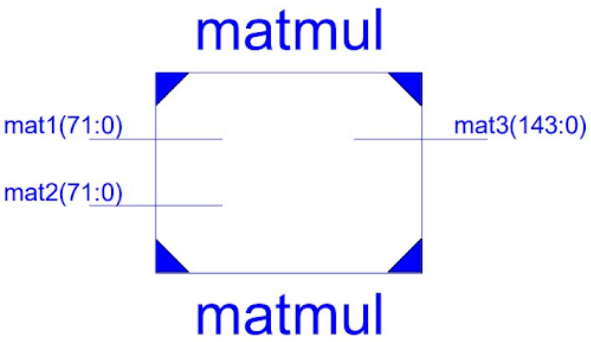

Figure 6.17: Black Box Diagram of 3x3 8- bit Matrix multiplier

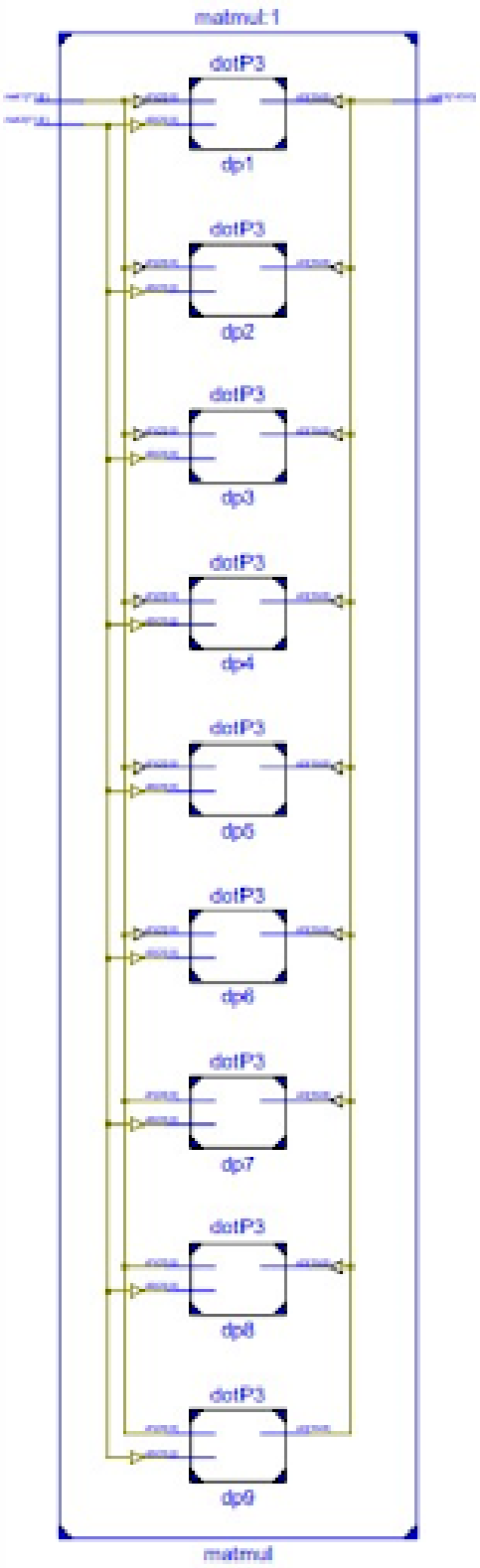

Figure 6.18: RTL Diagram of 8- bit Matrix multiplier

7. SIMULATION RESULT

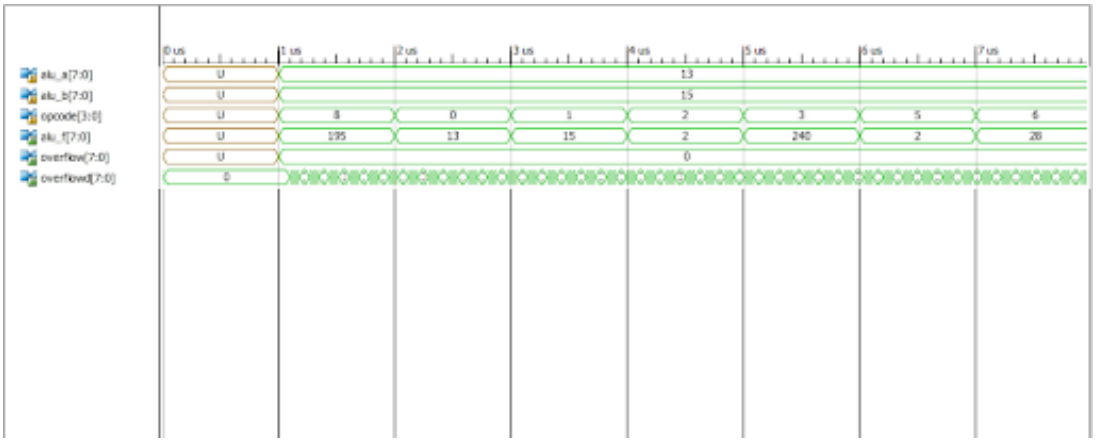

Figure 7.1: Different Operations for same Input Values

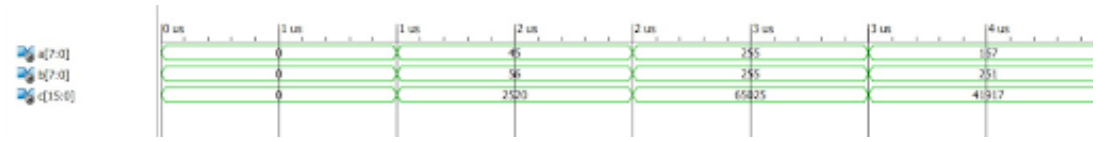

Figure 7.2: 8-bit Multiplication

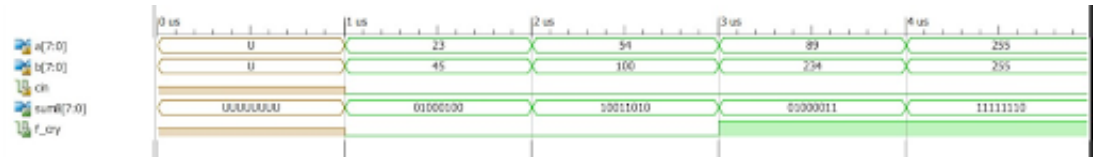

Figure 7.3: Addition (cin = 0)

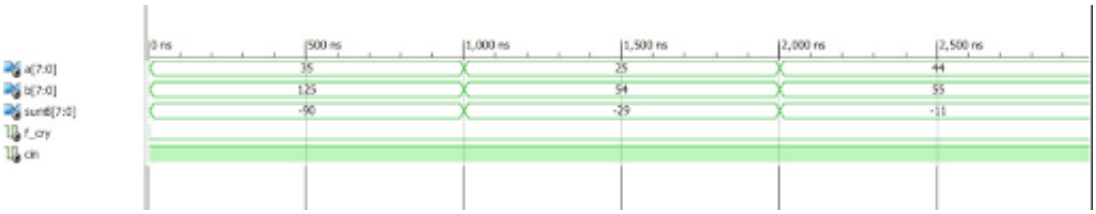

Figure 7.4: Subtraction (cin =1)

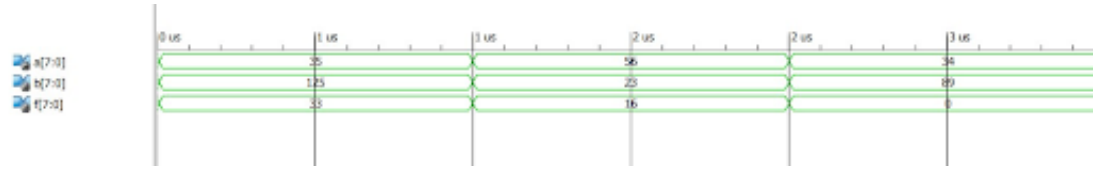

Figure 7.5: AND gate

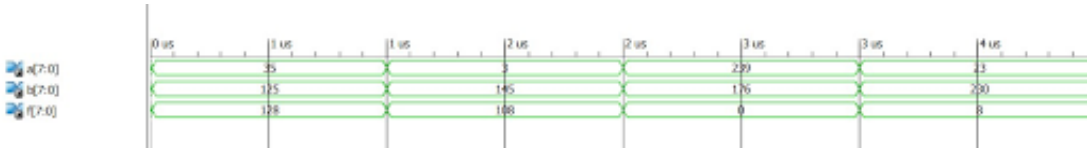

Figure 7.6: NOR gate

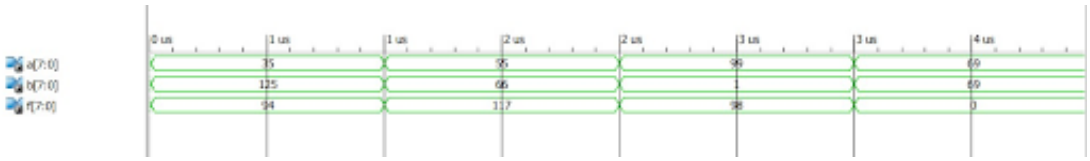

Figure 7.7: NAND gate

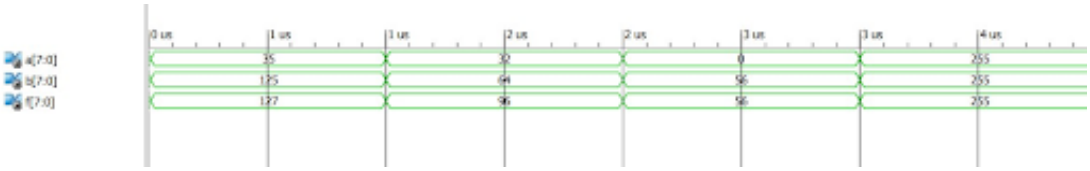

Figure 7.8: OR gate

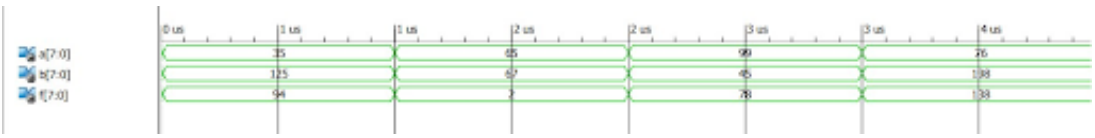

Figure 7.9: XOR gate

Figure 7.10: Matrix Multiplication

8. CONCLUSION

This paper presented a new idea to design an 8-bit ALU of a processor. This has been implemented in SPARTAN 3E-500 FPGA device. FPGA design offers greater design flexibility. The design is compact and is upgradable without any change in hardware. Here, the synthesis tool optimizes the design architecture of the FPGA for the ALU. Hence, additional features can be added to the existing design without any changes in hardware. The ALU is synthesized and simulated. The ALU of a processor is configured using each block designed.

The main advantage of the proposed 8- bit ALU design is the achievement of an increase in the speed of ALU operation so that the time consumption will be reduced. This is because of the fact that the proposed ALU performs the multiplication operation using ancient Vedic Mathematical Ideas which are surprisingly found to be faster than other Arithmetic Circuits that are implemented today. Thus, this project will make people aware about the existing ancient Vedic mathematical algorithms and their efficiency and speed.

Bibliography

[1] Bhoi N. Pradhan M. Dauda A.K. Barpanda N. *Control Unit Design of a 16-bit Processor Using VHDL*. International Journal of Advanced Research in Computer Science **and** Software Engineering, 2002.

[2] Deo A. Neupane M. Pun P. *Design of 4-bit ALU Using Logic Gates*. Nepal Engineering College, 2006.

[3] Borthakur R. Agarwal S. Takhar P. Rajpal P. *Design of 8-bit ALU And Implementing on Xilinx Vertex 4 FPGA*. Amity School of Engineering & Teachnology, 2011.

[4] M.H. Manas. *8 Bit Microprocessor Design Using VHDL*. 2007.

[5] S. Kilts. *Advanced FPGA Design*. A John Wiley & Sons, Inc., 2007.

[6] L. Redmond. *Design of a Teaching Instruction Set Processor in VHDL*. 2004.

[7] Douglas L.Perry. *Design of a Teaching Instruction Set Processor in VHDL*. Mcgraw Hill Education, 2002.