**Project report**

**on**

# Design And Implementation of 8*8 Vedic Multiplier Using Verilog in Xilinx Vivado

***Bachelor of Technology (Hons.)***

*in*

***Electronics & Communication Engineering***

*by*

**SIMRAN GUPTA (2022UGEC063)**

*Under the supervision of*

**Dr B.N.S MUNDA**

(Assistant Professor, Department of ECE, NIT Jamshedpur)

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**NATIONAL INSTITUTE OF TECHNOLOGY JAMSHEDPUR**

# CANDIDATE’S DECLARATION

We hereby declare that

a. The work contained in this report is original and has been done by us under the guidance of our supervisor *Dr B.N.S Munda,* Associate professor, Department of Electronics, NIT Jamshedpur.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. I have followed all the guidelines provided by the Institute in preparing the report.

d. I have conformed to the norms and guideline given in the Ethical Code of Conduct of my Institute.

e. Wherever I have used materials (data, theoretical analysis, figures and texts) from other sources, I have given due credit to them by citing them in the project report and giving their details in the reference. Further I have taken permission from the copyright owners of the sources, wherever necessary.

SIMRAN GUPTA
(2022UGEC063)

**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**

**National Institute of Technology Jamshedpur, Jamshedpur**

# CERTIFICATE

---

This is to certify that the project report entitled, "**Design and Implementation of 8*8 Vedic Multiplier using Verilog in Xilinx Vivado**" submitted by "***Simran Gupta (2022UGEC063)***" B.Tech (Hons.) students of ECE department of NIT Jamshedpur, India is a record of bona-fide Project Work carried out by them under my supervision. The work incorporated in this project has not been, to the best of my knowledge, submitted to any other university or Institute for the award or any degree or diploma.

# ACKNOWLEDGEMENT

This research work is one of the significant achievements in my life and is made possible because of the unending encouragement and motivation given by so many in every part of my life. It is immense pleasure to have this opportunity to express my gratitude and regards to them.

Firstly, I would like to express my gratitude and sincere thanks to **Associate Prof. Dr B.N.S MUNDA**, my supervisor, Department of Electronics and Communication Engineering for his esteemed supervision and guidance during the tenure of my project work.

Also, I would like to express my gratitude and respect to our faculty advisor **Dr B Bhowmik** and HOD of our Department **Dr (Prof) Dilip Kumar** for their support, feedback and guidance throughout my B. Tech course duration.

By

Simran Gupta

# CONTENTS

## Abstract

The need of high speed multiplier is increasing as the need of high speed processors and fast digital signal processing system are increasing which is a major source of power dissipation . Multipliers play an important role in today's digital signal processing and various other applications. Multiplier acts as a key block element in high speed arithmetic logic units, microcomputer, image processing etc. Multiplication process basically requires more hardware resources and computation time than addition and sub-traction process. In the recent years growth of the portable electronic is forcing the designers to optimize the existing design for better performance. Vedic mathematics have sixteen sutras but "Urdhva-Triyagbhyam" is mainly used. Vedic method for multiplication which is different from the process of normal multiplication is presented. This paper proposes the design of Vedic Multiplier using the techniques of Vedic Mathematics that have been modified to improve performance. Urdhva Triyagbhyam is the most efficient algorithm that gives minimum delay and reduce power consumption for multiplication for all types of numbers irrespective of their size. Simulation is done in Xilinx Vivado 2023.2 software using verilog.

**Software Used** : Xilinx Vivado

Supervisor

# Chapter 1

# INTRODUCTION

## 1.1 OVERVIEW

Multiplication lies at the core of many computational tasks, especially in digital signal processing, where efficiency is paramount. Traditional multiplication algorithms like Array, Booth, Bit serial, Carry save, Modified booth, and Wallace multipliers have served their purpose but come with their limitations. Array multiplier, for instance, offers speed but at the cost of extensive gate usage, making it economically impractical. Similarly, Wallace tree, despite its high-speed operation, faces layout challenges.

To address these limitations, the exploration of Vedic Mathematics presents a promising avenue. Rooted in ancient Indian scriptures, Vedic Mathematics offers a rich tapestry of mathematical principles encapsulated in sixteen Sutras. These principles, discovered and reconstructed by Swami Bharati Krishna Tirthaji Maharaja, provide a systematic and efficient approach to arithmetic operations.

The integration of Vedic Mathematics into multiplier design holds significant potential for enhancing computational efficiency. By leveraging techniques such as "Urdhva Tiryagbhyam" (vertically and crosswise), the aim is to minimize computational time while maintaining or even improving other essential design aspects such as speed, power consumption, symmetry, and area.

In this report, we delve into the realm of high-speed multiplier design, drawing inspiration from Vedic Mathematics. Through the synthesis of ancient wisdom and modern technology, we propose a design using Verilog in Xilinx Vivado.

Furthermore, our report aims to serve as a testament to the relevance and applicability of ancient knowledge in contemporary engineering endeavours. Through our exploration of Vedic Mathematics in multiplier design, we hope to inspire further research and innovation, ultimately contributing to the advancement of digital design methodologies and computational techniques.

## 1.2 MOTIVATION

The need of processor is increasing in this high edging technological society is increasing day by day . A conventional processor requires substantially more hardware resources and processing time in the multiplication operation, rather than addition and subtraction. Thus, optimizing multiplier efficiency is paramount to overall system performance and energy conservation.

- **Reduce the number of steps in multiplication process:** Traditional multiplication algorithms often involve multiple steps to compute the product of two numbers. By integrating Vedic Mathematics principles into the design, such as the Urdhva-Tiryagbhyam sutra, we aim to streamline the multiplication process. This sutra emphasizes vertical and crosswise operations, allowing for a more direct and efficient computation of the product. By reducing the number of intermediate steps, we can achieve faster multiplication with fewer computational overheads.
- **Reduce computational delay:** Computational delay refers to the time taken for a system to produce an output after receiving inputs. Traditional multiplication algorithms may exhibit significant delay due to complex logic and sequential processing. By leveraging Vedic Mathematics techniques, which emphasize simplicity and efficiency, we can minimize the computational delay. For example, by using parallel processing and optimized algorithms, we can expedite the multiplication process, thereby reducing overall delay.
- **Reduce power consumption:** Power consumption is a critical consideration in modern digital designs, especially for portable and battery-operated devices. Traditional multiplication algorithms may consume significant power due to the utilization of complex logic and high gate counts. By optimizing the multiplier architecture using Vedic Mathematics principles, we can reduce power consumption. For instance, by minimizing unnecessary switching activities and optimizing the routing of signals, we can achieve lower power consumption without compromising performance.
- **Reduce memory utilization:** Memory utilization refers to the amount of memory resources required to store intermediate data during the computation process. Traditional multiplication algorithms may necessitate large memory buffers to accommodate

intermediate results, leading to increased memory utilization. By employing efficient data handling techniques inspired by Vedic Mathematics, such as dynamic allocation and reuse of resources, we can reduce memory requirements.

## 1.3 LITERATURE REVIEW

2. **Dr Savita Sonoli, Sahana** Desai "Design and Implementation of 4x4 Vedic Multiplier using Cadence". IJERRT ISSN: 2278-0181 Vol. 9 Issue 07, July-2020
   An analysis of the research paper reveals that the authors have effectively addressed the objectives of reducing delay, enhancing speed, and lowering power consumption in multiplier designs. The use of Vedic mathematics and techniques such as the Urdhva-Triyagbhyam sutra demonstrates innovative approaches to multiplier optimization.
   However, the paper could benefit from more detailed discussions on the experimental methodology, performance metrics, and potential drawbacks or limitations of the proposed design. Additionally, providing insights into the practical implementation challenges and future research directions would further enrich the paper's content.
   Overall, the paper provides valuable insights into the application of Vedic Mathematics in high-speed multiplier design and presents a comparative analysis of different architectural approaches.

3. **Samiksha Dhole, Sayali Sembadkar,Tirupati Yadav, Dr Prasheel Thakre** "Design and FPGA Implementation of 4x4 Vedic Multiplier using Different Architectures."
   IJERRT ISSN: 2278-0181 Vol. 6 Issue 04, April-2017.The authors present the Vedic multiplication method for 2x2 bit and 4x4 bit numbers, along with their respective block diagrams and implementation details. They discuss various

architectures for the 4x4 Vedic multiplier, including those using Ripple Carry Adder, Look Ahead Carry Adder, and Carry Save Adder. The paper includes detailed explanations of each architecture, VHDL code. snippets, and synthesis analysis results for different FPGA implementations.

However, there are some limitations and drawbacks in the paper. Firstly, while the research focuses on reducing delay, it does not extensively discuss power consumption or area efficiency, which are also crucial factors in hardware design. Secondly, the paper lacks detailed discussions on the trade-offs between different architectures and their suitability for specific applications. Additionally, it would be beneficial to include a discussion on the scalability of the proposed designs for larger bit-width multipliers.

# Chapter 2

# THEORETICAL STUDY

A conventional process of multiplication require more steps , time delay and power consumption so to overcome these challenges we introduce vedic mathematics that has sixteen sutras among them we mainly use “urdhva Tiryagbhyam” also known as “vertically and crosswise” it allows for the calculation of products and sums simultaneously in a single step, making it a highly efficient method for multiplication.

## 2.1 Urdhva-Tiryagbhyam Sutra —

The Urdhva-Tiryagbhyam Sutra, also known as "Vertically and Crosswise," is a general multiplication formula in Vedic Mathematics. It allows for the calculation of products and sums simultaneously in a single step, making it a highly efficient method for multiplication.
In traditional multiplication, each digit of the multiplier is multiplied by each digit of the multiplicand, and the results are then added together to obtain the final product. This process involves multiple sequential steps, leading to higher timing delays, especially as the number of bits increases. However, the Vedic approach using the Urdhva-Tiryagbhyam Sutra operates efficiently. It performs both vertical and crosswise multiplications in parallel, enabling faster computation.

### Steps of working:

Vertical Multiplication: In this step, the corresponding digits of the multiplicand and multiplier are multiplied vertically. For example, in a 4x4 multiplication, the least significant bits (LSBs) of both numbers are multiplied to produce the LSB of the result.
Crosswise Multiplication: In this step, the LSB of one number is multiplied by the other digits of the second number and vice versa. These crosswise products are then added together to obtain partial sums.

Combining Results: The results from vertical and crosswise multiplication are combined to produce the final product.

**Here the example to illustrate the sutra by multiplying 32 by 12**

- We multiply 2 of the multiplier and 2 of the multiplicand vertically and get 4 as the answer. This becomes the right-hand most part of the answer

- We then multiply cross-wise i.e 3 and 2, and 2 and 1, add the two (3*2)+(1*2), get 8 as their sum and set it down as the middle part of the answer

- We multiply left-hand most digit 1 of the multiplier vertically by the left-hand most digit 0 of the multiplicand, get their product (1*1=1) and set it down as the left-hand most part of the answer. Thus 32*12 = 384

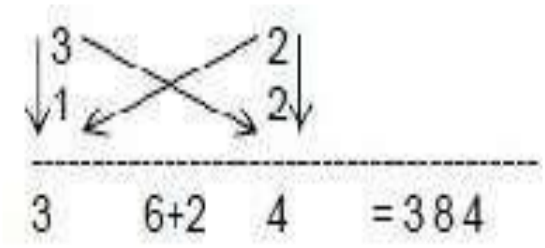


Fig 2.1: Multiplication of 32*12 using Urdhva-tirygabhyam method

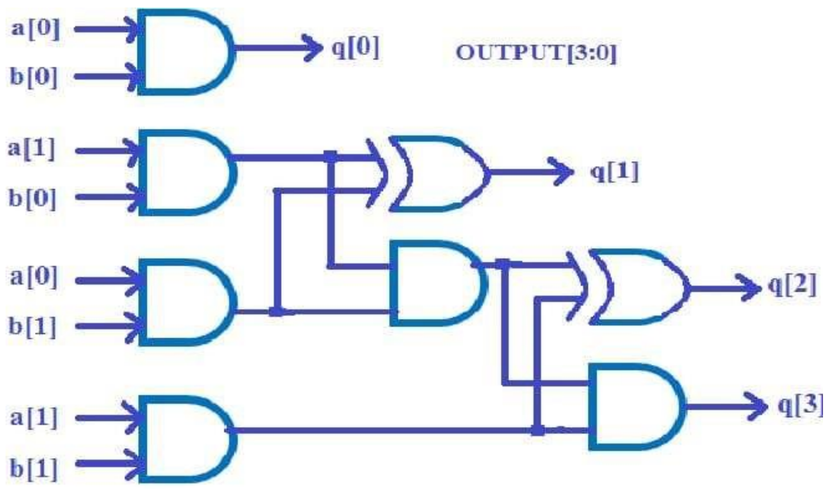
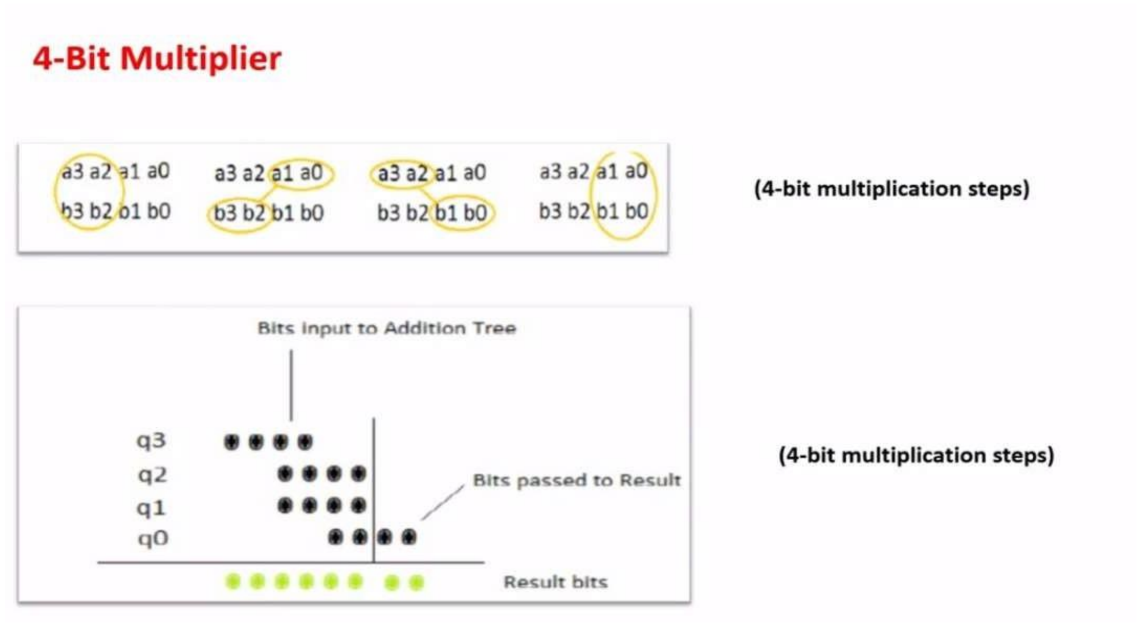


Fig 2.2: Circuit diagram of 2-bit Vedic multiplier



4-Bit Multiplier
a3 a2 a1 a0
b3 b2 b1 b0
(4-bit multiplication steps)
Bits input to Addition Tree
q3
q2
q1
q0
Bits passed to Result
Result bits
(4-bit multiplication steps)

Fig 2.3: 4-bit Vedic multiplication process

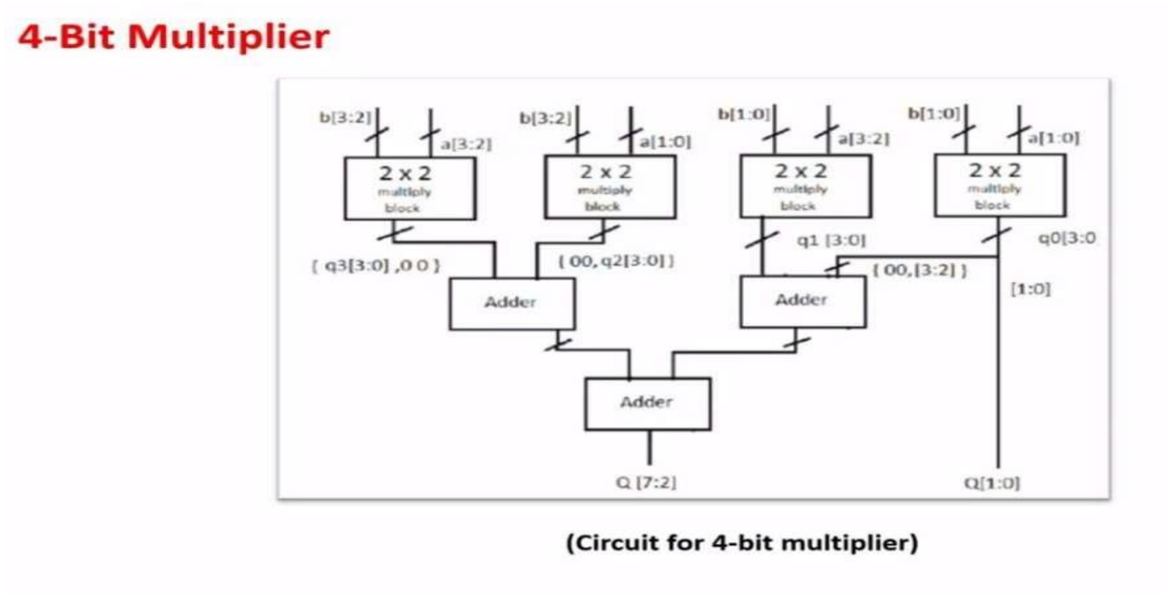


Fig 2.4: Circuit diagram of 4-bit Vedic multiplier

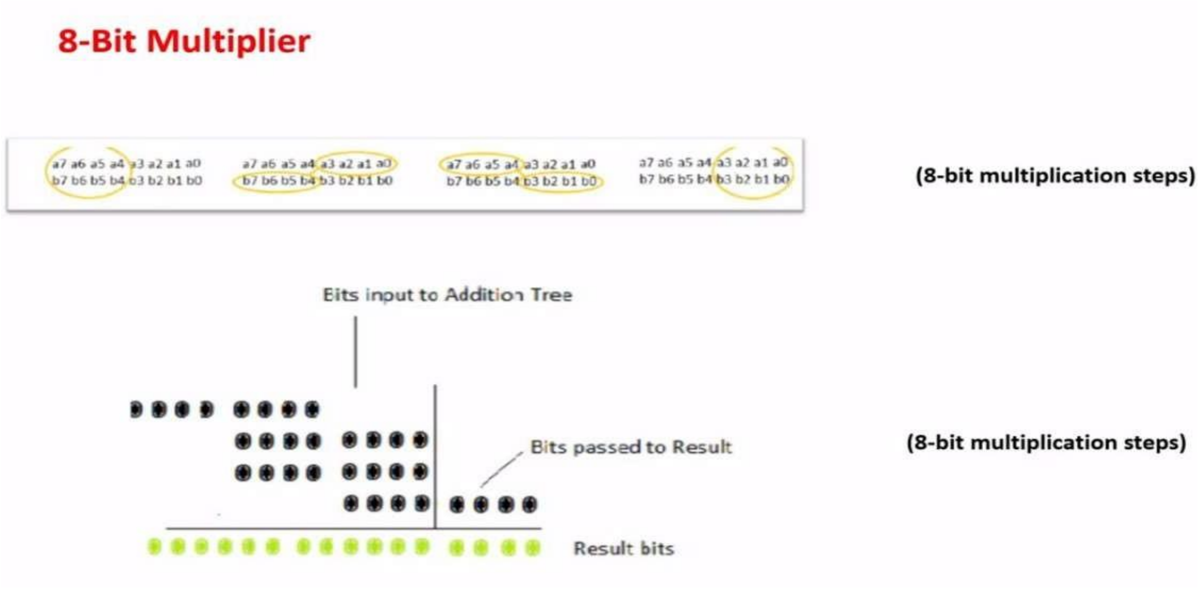
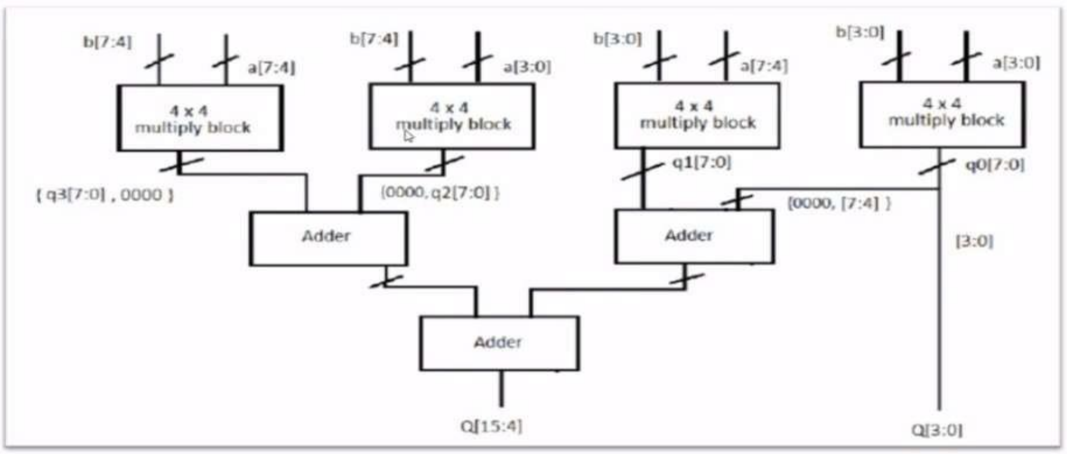


Fig 2.5: 8-bit Vedic multiplication process

**8-Bit Multiplier**



**(Circuit for 8-bit multiplier)**

Fig 2.6: Circuit diagram of 8-bit Vedic multiplier

# Chapter 3

# EXPERIMENTAL STUDY AND WORK DONE

## 3.1 : Vivado -Xilinx Software —

Vivado, developed by Xilinx, is a comprehensive design environment for FPGA (Field-Programmable Gate Array) development. It provides tools and features to design, implement, and verify digital circuits efficiently.

**In this , we are implementing how Vivado can be utilized to simulate the design of an 8-bit multiplier**.

## 3.2 VERILOG FILE

Open Xilinx Vivado and create a new project. Click on "Create Project" from the Vivado welcome screen or File menu. Specify the project name, location, and project directory. Click "Next" and follow the wizard to specify the project settings. Add the Verilog file to the project: In the Flow Navigator panel, click on "Add Sources" then select "Add or create design sources" and click "Next". Browse and select the Verilog file you created earlier. Click "Finish" to add the file to the project.

## DESCRIPTION OF VERILOG CODE

Here's a step-by-step breakdown of the Verilog code for the 8-bit Vedic multiplier testbench:

Module Declaration (module test_vedic_8;):

This declares the testbench module named test_vedic_8.

Input and Output Declarations:

reg [7:0] a; declares an 8-bit register named a.

reg [7:0] b; declares an 8-bit register named b.

wire [15:0] c; declares a 16-bit wire named c to capture the output of the Vedic multiplier.

Instantiation of Unit Under Test (UUT):

vedic_8X8 uut (.a(a),.b(b), .c(c)); instantiates the 8-bit Vedic multiplier module (vedic_8X8) with inputs a and b, and output c.

Initial Block:

initial begin marks the beginning of the initial block, which contains the initial simulation behavior.

$monitor($time," a= %b, b=%b, --- c= %b\n", a, b, c); sets up monitoring of the values of a, b, and c and prints them whenever they change.

Test Cases:

a = 0; b = 0; #100;: Sets a and b to 0 and waits for 100 time units.

a = 8'd255; b = 8'd255; #100;: Sets a and b to 255 and waits for 100 time units.

a = 8'd5; b = 8'd3; #100;: Sets a to 5 and b to 3 and waits for 100 time units.

a = 8'd4; b = 8'd2; #100;: Sets a to 4 and b to 2 and waits for 100 time units.

a = 8'd2; b = 8'd2; #100;: Sets a and b to 2 and waits for 100 time units.

a = 8'd6; b = 8'd8; #100;: Sets a to 6 and b to 8 and waits for 100 time units.

End of Initial Block:

end marks the end of the initial block.

End of Module:

endmodule marks the end of the testbench module.

## 3.3 SYNTHESIS AND VIEW SIMULATION:

- Before synthesis and implementation, it's a good practice to simulate the design to ensure its functionality. Launch the simulation tool in Vivado and simulate the testbench (**test_vedic_8.v**) to verify the functionality of the 8x8 Vedic multiplier. Once simulation results are satisfactory, proceed with synthesis. Vivado will synthesize the Verilog code and generate a gate-level netlist

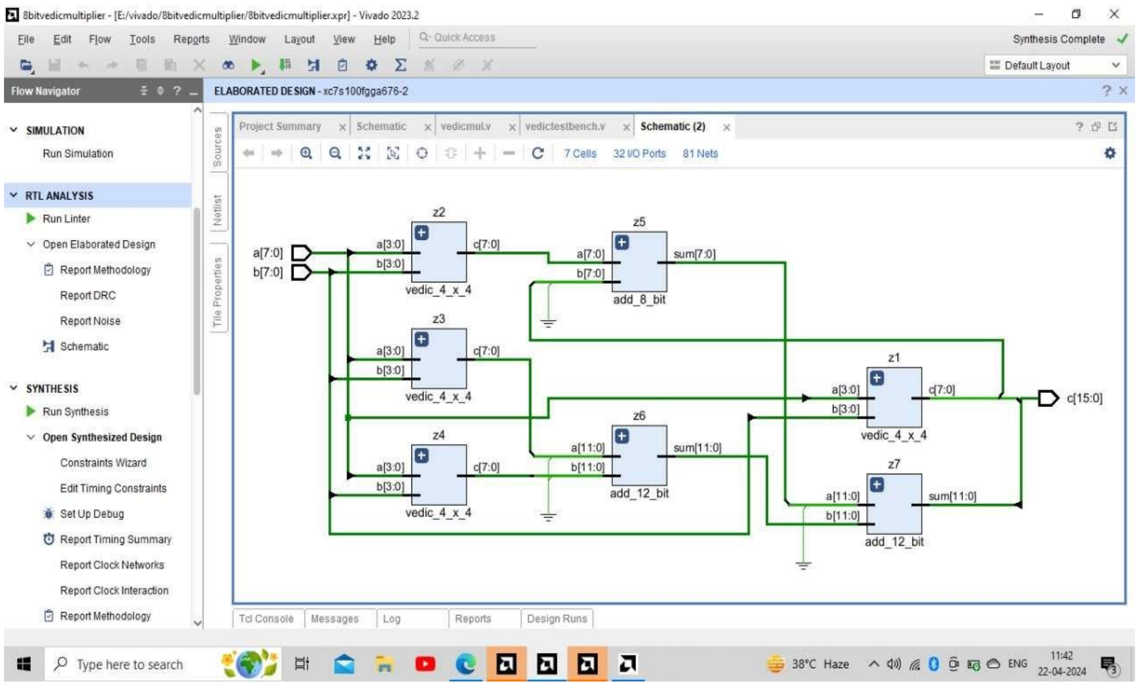


Fig 2.7: Schematic View of 8-bit multiplication on Xilinx Vivado

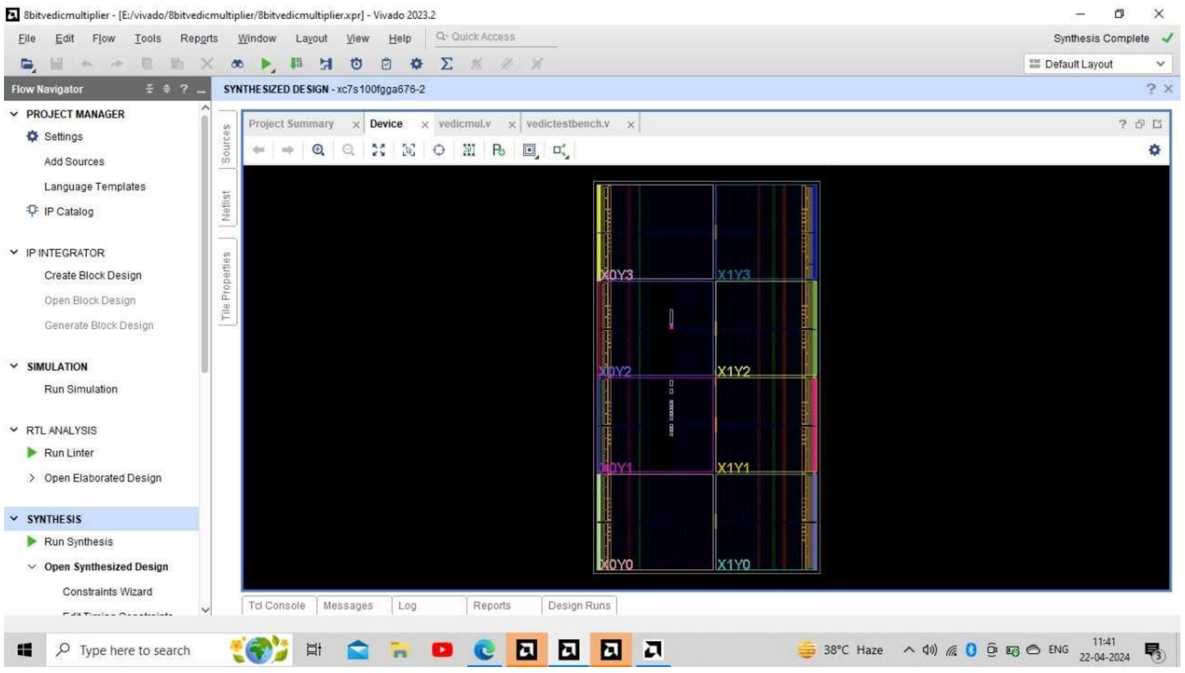


Fig 2.8: Synthesized view of 8-bit Vedic multiplication

## 3.4 VIEW RESULT AND IMPLEMENTATION

After synthesis, proceed with implementation. The behaviour of the testbench and the results for the given test cases:

**Test Case 1: a = 255, b = 255**

This test case is designed to test the multiplication of two maximum 8-bit values.

Result:

Since both a and b are set to their maximum value (255), the product should be 255 * 255 = 65025, which is represented as a 16-bit binary number 1111111111111111.

Therefore, the output (c) is 1111111111111111.

**Test Case 2: a = 5, b = 3**

This test case tests multiplication of relatively small numbers.

Result:

The product of 5 and 3 is 15, represented in binary as 0000000000001111.

Therefore, the output is 0000000000001111.

**Test Case 3: a = 4, b = 2**

This test case also involves small numbers.

Result:

The product of 4 and 2 is 8, represented in binary as 0000000000001000.

Therefore, the output is 0000000000001000.

**Test Case 4: a = 2, b = 2**

This test case checks for multiplication of equal numbers.

Result:

The product of 2 and 2 is 4, represented in binary as 0000000000000100.

Therefore, the output is 0000000000000100.

**Test Case 5: a = 6, b = 8**

This test case involves multiplication of two relatively larger numbers.

Result:

The product of 6 and 8 is 48, represented in binary as 0000000000110000.

Therefore, the output is 0000000000110000.

These test cases cover a range of scenarios, including edge cases, small numbers, and larger numbers, ensuring thorough testing of the 8-bit Vedic multiplier.

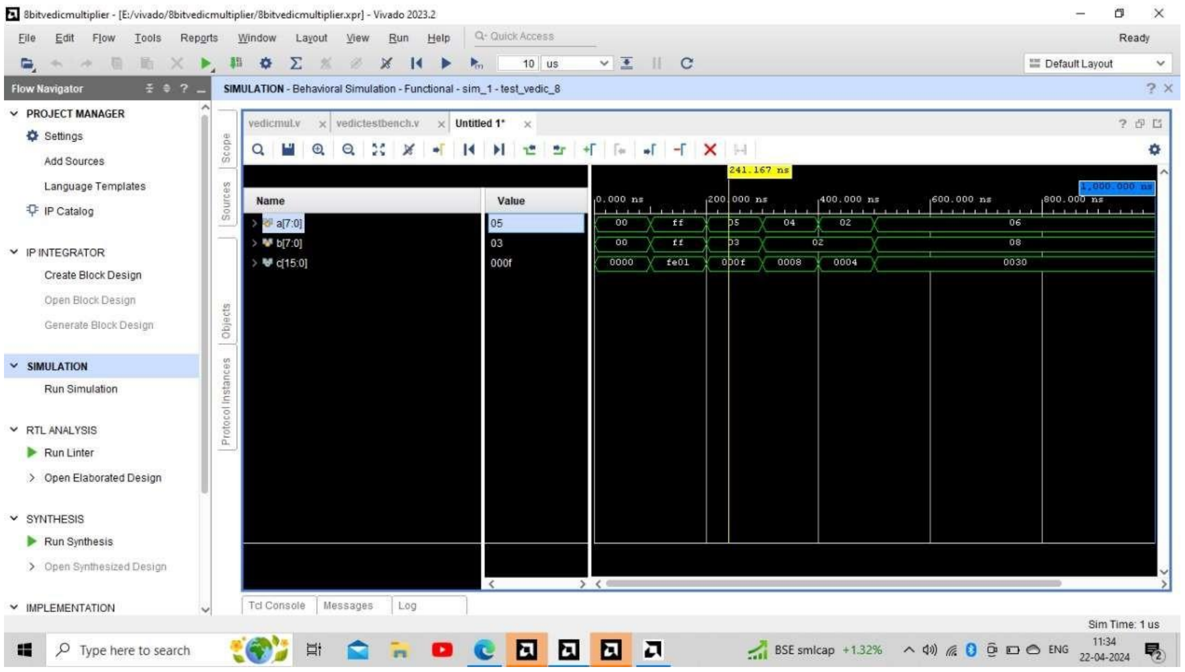

8bitvedicmultiplier - [E:/vivado/8bitvedicmultiplier/8bitvedicmultiplier.xpr] - Vivado 2023.2
SIMULATION - Behavioral Simulation - Functional - sim_1 - test_vedic_8
Name
Value
a[7:0]
05
b[7:0]
03
c[15:0]
000f
241.167 ns
Sim Time: 1 us

Fig 2.9: Behavioural simulation on Xilinx Vivado

# Chapter 4

# Conclusion and Scope of future work

## 4.1 CONCLUSION

Our project addresses the demand for high-speed multipliers and also contributes to the realm of efficient multiplication by leveraging Vedic Mathematics.

The implementation of an 8-bit multiplier using Xilinx Vivado software demonstrates the effectiveness of Vedic mathematics techniques in digital design. Through the use of the Urdhva-Tiryagbhyam Sutra and various architectures, we aimed to achieve high-speed multiplication with reduced delay and power consumption.

Our simulations in Vivado showed that the Vedic multiplier using different architectures resulted in varying combinational delays. Among the architectures tested, the Vedic Multiplier for 8x8 Bit using vedic method exhibited the lowest combinational delay, indicating its potential for high-speed multiplication.

The results confirm the advantages of using Vedic mathematics in digital design. By leveraging ancient mathematical principles, we were able to create efficient multipliers that outperform conventional methods in terms of speed and resource utilization.

As the number of steps in multiplication is reduced, delay is reduced and hence speed of the multiplier increases. Vedic Algorithm like Urdhav Tiryagbhyam is best suited for high speed application

.Ease of implementation of the design shows that complex modules for DSP and image processing can be simplified using Vedic algorithm.

## 4.2 SCOPE OF FUTURE WORK

In future we will do simulations and synthesis analyses in Vivado showing the Vedic multiplier using different architectures resulting in varying combinational delays.

Vivado will be employed to perform place and route optimization of the synthesized design onto the targeted FPGA device. This optimization step is crucial for achieving optimal performance and resource utilization of the multiplier design.

Overall, In future our aim is to deliver a fully functional 8x8 Vedic multiplier implementation on FPGA, ensuring efficient resource utilization, reliable performance, and adherence to project requirements.

## 4.3 REFERENCES

| | |
|---|---|
| 1.Dr. Savita Sonoli, Sahana Desai IJERRT ISSN: 2278-0181 Vol. 9 Issue 07, July-2020 | Design and Implementation of 4x4 Vedic Multiplier using Cadence". |
| 2.Samiksha Dhole, Sayali Sembadkar,Tirupati Yadav, Dr Prasheel Thakre IJERRT ISSN: 2278-0181 Vol. 6 Issue 04, April-2017. | Design and FPGA Implementation of 4x4 Vedic Multiplier using Different Architectures. |
| 3. Jagadguru Swami Sri Bharati Krishna Tirthaji Maharaja | Vedic Mathematics or Sixteen Simple Mathematical formula form the veda ,delhi (1965) |
| 4. Paras Gulati,Harsh Yadav, Manoj Kumar Taleja 2016 International Conference on Computing, Communication and Automation (ICCCA) IEEE . | Implementation of an efficient Multiplier Using the Vedic Multiplication Algorithm |
| 5. : M. M. Mano and M. D. Ciletti, 5th Ed., Pearson Education | Digital Logic and Computer Design . |
| 6. S. Palnitkar, Pearson, 2nd Ed, 2003 | Verilog HDL: A Guide to Digital Design and Synthesis |
| 7.Verilog Code and Testbench cases | https://github.com/simrangupta29/minorproject_8bitvedicmultiplier |

## 4.3 APPENDIX

Verilog code for 8-bit Vedic multiplier

```
module ha(a,b,sum,carry);
input a,b;
output sum,carry;
xor(sum,a,b);
and(carry,a,b);
endmodule

module add_4_bit (a,b,sum);
input [3:0] a,b;
output  [3:0]sum;
assign sum=a+b;
endmodule

module add_6_bit (a,b,sum);
input [5:0] a,b;
output [5:0] sum;
assign sum = a+b;
endmodule

module add_8_bit (a,b,sum);
input[7:0] a,b;
output[7:0] sum;
assign sum = a+b;
endmodule

module add_12_bit (a,b,sum);
input[11:0] a,b;
output[11:0] sum;
assign sum = a+b;
endmodule



module vedic_2_x_2(a,b,c);
input [1:0]a;
input [1:0]b;
output [3:0]c;
wire [3:0]c;
wire [3:0]temp;
```

```
assign c[0]=a[0]&b[0];
assign temp[0]=a[1]&b[0];
assign temp[1]=a[0]&b[1];
assign temp[2]=a[1]&b[1];
ha z1(temp[0],temp[1],c[1],temp[3]);
ha z2(temp[2],temp[3],c[2],c[3]);
endmodule

module vedic_4_x_4(a,b,c);
input [3:0]a;
input [3:0]b;
output [7:0]c;
wire [3:0]q0;
wire [3:0]q1;
wire [3:0]q2;
wire [3:0]q3;
wire [7:0]c;
wire [3:0]temp1;
wire [5:0]temp2;
wire [5:0]temp3;
wire [5:0]temp4;
wire [3:0]q4;
wire [5:0]q5;
wire [5:0]q6;

vedic_2_x_2 z1(a[1:0],b[1:0],q0[3:0]);
vedic_2_x_2 z2(a[3:2],b[1:0],q1[3:0]);
vedic_2_x_2 z3(a[1:0],b[3:2],q2[3:0]);
vedic_2_x_2 z4(a[3:2],b[3:2],q3[3:0]);

assign temp1 ={2'b0,q0[3:2]};
add_4_bit z5(q1[3:0],temp1,q4);
assign temp2 ={2'b0,q2[3:0]};
assign temp3 ={q3[3:0],2'b0};
add_6_bit z6(temp2,temp3,q5);

assign temp4={2'b0,q4[3:0]};
add_6_bit z7(temp4,q5,q6);
assign c[1:0]=q0[1:0];
assign c[7:2]=q6[5:0];
endmodule

module vedic_8X8(a,b,c);

input [7:0]a;
input [7:0]b;
output [15:0]c;
```

```
wire [15:0]q0;
wire [15:0]q1;
wire [15:0]q2;
wire [15:0]q3;
wire [15:0]c;
wire [7:0]temp1;
wire [11:0]temp2;
wire [11:0]temp3;
wire [11:0]temp4;
wire [7:0]q4;
wire [11:0]q5;
wire [11:0]q6;

vedic_4_x_4 z1(a[3:0],b[3:0],q0[15:0]);
vedic_4_x_4 z2(a[7:4],b[3:0],q1[15:0]);
vedic_4_x_4 z3(a[3:0],b[7:4],q2[15:0]);
vedic_4_x_4 z4(a[7:4],b[7:4],q3[15:0]);

assign temp1 ={4'b0,q0[7:4]};
add_8_bit z5(q1[7:0],temp1,q4);
assign temp2 ={4'b0,q2[7:0]};
assign temp3 ={q3[7:0],4'b0};
add_12_bit z6(temp2,temp3,q5);
assign temp4={4'b0,q4[7:0]};

add_12_bit z7(temp4,q5,q6);

assign c[3:0]=q0[3:0];
assign c[15:4]=q6[11:0];
endmodule
```