Investigating Performance Improvement for Montgomery Modular Operations using Vedic Multipliers

Prasanna Balaji – pbalaji@kth.se
Sebin Shaji Philip – sebin@kth.se

December 13,2018

Abstract

Efficient hardware implementations of cryptographic algorithms have been of much interest recent years where speed and performance are a crucial factor. All the famous cryptographic algorithms use modular operations (Montgomery) to avoid direct multiplication and division of very large numbers. In this paper we investigate the performance improvement of Montgomery modular operations by replacing normal multipliers with Vedic multipliers in FPGA.

Keywords: Cryptography, Montgomery modular operations, Vedic multiplication, Booths multiplier, FPGA.

TABLE OF CONTENTS

List of Figures

List of Tables

1 Introduction

1.1 Background

Cryptography involves the encryption and secure transmission of data to ensure information security. This is usually accomplished using ciphers. It is commonly used for authentication, data integrity, confidentiality and other similar applications. It is becoming increasingly popular to see cryptography in a growing number of industries, from finance and banking to electronic commerce to communication, etc.

Public key cryptographic systems are ubiquitous. A wide number of public-key algorithms, such as RSA (Rivest-Shamir-Adleman), AES (Advanced Encryption Standard), and Elliptic Curve cryptography makes use of complex mathematical operations and large prime numbers. Prime numbers can be determined by tests such as the Rabin-Miller primality testing algorithm. Modular exponentiation forms the base of a lot of these prime number generators and crypto systems. Montgomery modular operations, Montgomery multiplication in specific is the fastest available means of implementing this exponentiation, and hence are widely used in cryptographic applications.

Montgomery multiplication involves computing the result from numbers which are converted into their Montgomery forms. Montgomery forms of numbers are computed by representing a as (ar mod N). Given x, y and z, the result from this multiplication is xy mod z. The modulus operation is optimized by achieving modulus division through a series of additions and smaller divisions (power of 2) which also makes hardware implementations feasible.

The necessary computations can be further optimized by utilizing a fast system of mathematical reasoning called vedic maths. Vedic mathematics greatly simplifies the steps involved in mathematical operations by employing a number of shortcuts. It is said to be the fastest way to do mathematics[??]. The core of vedic mathematics is its 16 sutras which attribute a characteristic to different groups of numbers. It brings down complex calculations to a mere two or three steps and is extremely effective for computing multiplications and divisions.

In our proposed research, we look at bringing about a performance improvement in the widely used montgomery modular operations, by making use of vedic mathematics. We plan to implement a vedic multiplier on FPGA and correlate its output to an FPGA implementation of montgomery multiplication to investigate the potential benefit.

1.2 Literature survey

Basic ALU architecture consists of basic arithmetic building blocks like adder, multiplier, floating point unit, shifter, etc. Performance improvement in one of these basic logic units will considerably contribute towards the overall system performance. We are interested particularly in multipliers as there are a large number of repeated multiply operations involved in Cryptographic algorithms.

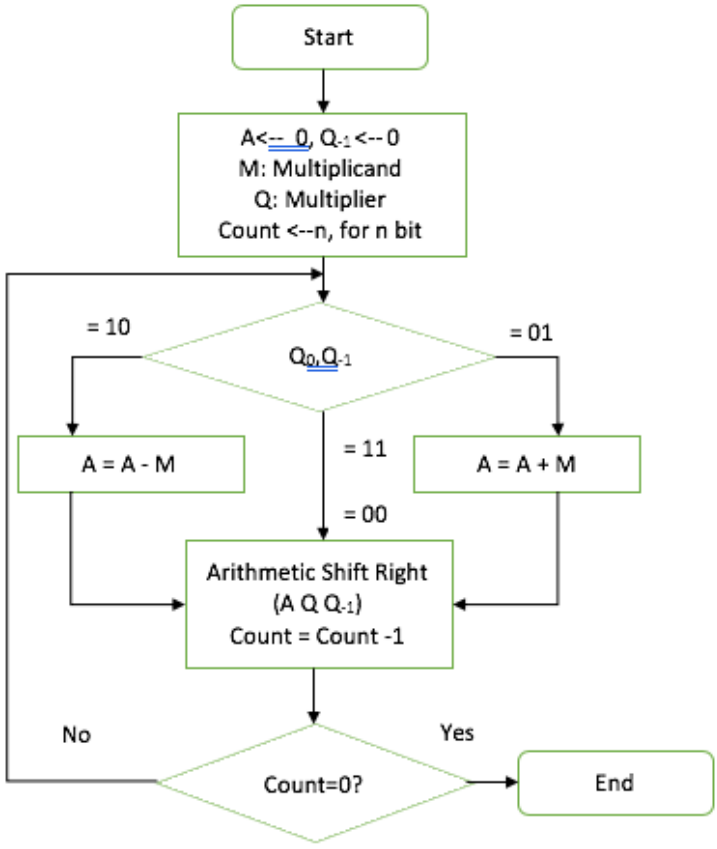

Figure I: Booths multiplication block diagram

The most common implementations of multipliers are based on booths [4] and array multiplier algorithms. It's evident that booths radix 4 multiplier is found to be better among radix-2 booths, radix-4 booths and array multiplier considering a hardware

implementation [3]. Booths multiplier gives a procedure for multiplying binary integers in signed 2's compliment [4]. Booths algorithm requires examining of the multiplier bits, and shifting of the partial product. The multiplicand may be added to or subtracted from the partial product prior to the shifting based on the bits Q_0Q_{-1} as shown in the Figure I. By repeating these steps for n-cycles we would arrive at a result for n-bit operands. The comparative study done in this literature [3] implements and compares different traditional multipliers like booths radix-2, radix-4, array multiplier and concludes that parallel implementation of booths radix-4 is more efficient compared to others.

Our special interest lies in exploring the Vedic multiplier for FPGA hardware architecture [5] and comparing its performance analysis with booths multiplier.

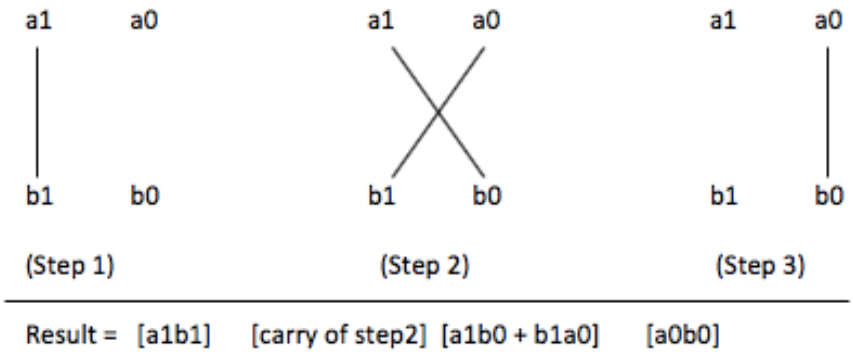

Figure II: Vedic 2-bit multiplication algorithm

Vedic multiplication is based on the ancient Indian Vedic mathematics [6,7,] sutra(formula) called Urdhva Tiryakbhyam (Vertically and Cross wise) which is traditionally used for decimal system in ancient India. This concept is extended for 4bit and 8bit binary numbers [5] in FPGA and shown significant performance improvement. The binary product of two 2-bit umbers a1a0 and b1b0 is obtained from crosswise and vertical multiplication as shown in Figure II. a0b0(1^{st} bit) and a1b1(4^{th} bit) are the vertical products and a1b0+b1a0(2^{nd} bit) is the crosswise product. The carry from crosswise product is taken as 3^{rd} bit of the final result. The same logic is implemented using logic gates as shown in Figure III.

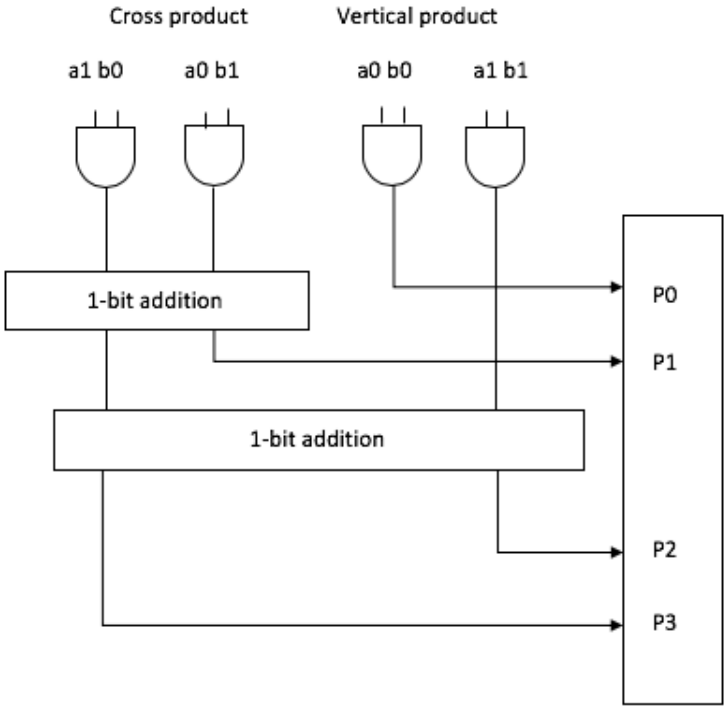

Figure III: Vedic 2-bit multiplier block

A 4-bit Vedic multiplier uses the same crosswise and vertical concept which is used for 2-bit and finds the 8-bit product. The 2-bit multiplier module designed in the previous step is reused to find the product for 4-bit multiplier and thus overall efficiency is improved. Four 2-bit multipliers and three 4-bit Full adders are used for constructing 4-bit multiplier as described in the literature [5]. Again a total of four 4-bit multipliers and three 8-bit Full adders are used for the construction of 8-bit multiplier blocks [5].

Montgomery multiplication is a fast method to compute modular products, typically used when there are a large number of multiplications to be performed in a row as it does not use trial division [8].

Montgomery multiplication performs the following operations [8]:

1. Conversion of operands to a modular form called Montgomery representation. (a = aR mod N).
    a. Choose suitable N
    b. Calculate Number of bits needed for binary representation of N. Call it as 'x'.
    c. Calculate R = 2^x ensuring that it is coprime to N [8].
    d. Calculate N^{-1} mod R, usually using the extended Euclidian algorithm.
2. Multiplying the Montgomery forms of the operands. (a x b corresponds to abR^2 mod N)

3. First Montgomery reduction to obtain the Montgomery form of the product. (abR mod N
4. Performing the reduction operation again to convert out of Montgomery form. (ab mod N)

Previous work indicates that that have been effective implementations of Montgomery multiplication in the past, for instance using a systolic array [10], a high radix variant [8] or using a redundant radix number system [13].

2 Research questions, hypotheses

It needs to be closely investigated if the potential performance improvement that could be achieved by using Vedic multipliers for Montgomery operations is beneficial. This project proposes to compare the performance of a booth multiplier against a multiplier incorporating Vedic mathematics. This is done through research, implementation and real-time simulation of the aforementioned methods which will be elaborated in section 4. The results obtained from this comparison will be further used to determine the impact of a high-performance multiplier in Montgomery multiplication. Theory and quantitative analysis used in section 5 will draw a conclusion towards the benefits of the proposed approach.

The experimental outcomes and simulated results along with the theoretical specifications relevant to their use case should be an effective guide for future research.

3 Research Methodology

This project involves a lot of computation and experiments and the Quantitative research methodology and model is best suited. A Qualitative style of research would not work well for the aforementioned project as it does not rely on interpretative investigation, and instead uses hard data to arrive to concrete conclusions. The research methodology follows the onion model [12] and is layered into different subsections as subsequently mentioned. The time horizon [12] for these methodologies is cross-sectional as the results are expected to stay the same regardless of which period in time it is investigated, provided all other involved factors remain the same.

3.1 Research Methods:

This project primarily follows an experimental style of research [11] as it involves modification of one component of an existing system to investigate how the result is

affected and determine the impact on system performance. Here, traditional multipliers are modified using VHDL to incorporate Vedic mathematics and the results of this improvement are computed. Since information from real work is also used, such as the existing Montgomery multiplication algorithm, there is also an applied research [11] undertone in this project. Moreover, knowledge derived from experiments is also quantitatively analysed to evaluate performance, thus making it follow an empirical research model [11]. Concrete data indicating the performance improvement in multipliers is applied to the multiplications in the Montgomery modular operation to hypothesize the potential performance enhancements in that algorithm. This combination of complementary research methods is best suited for this project to obtain the expected results.

Other research methods such as Non-experimental, descriptive, analytical, fundamental, conceptual, Quasi-Experimental, correlational and historical are largely qualitative in nature and aren't used for the purpose of this research.

3.2 Research Approach:

The deductive approach [11], or deductive reasoning as it is also known, forms the basis of this project. This research approach ensures the correlation between the obtained data and the theory in consideration by either verifying or falsifying hypotheses. Here, data about the contribution of Vedic mathematics in improving multiplier performance is deduced and used to determine if it could be used to bring about a significant enhancement in the efficiency of the Montgomery multiplication algorithm. The hypothesis in question, that will be verified by this research approach is regarding the potential improvement that a modified multiplier is capable of inducing. The only possible outcomes are true or false depending on whether the hypothesis is verified or falsified. The outcome is based on the data that is collected and is a generalization that is would be valid in most conditions.

The other research approaches would not fare well with this project as the inductive approach relies on qualitative data, and the abductive approach begins with an incomplete set of data or observations, both of which aren't applicable.

3.3 Research Strategy

The research strategy followed in this project is experimental [11]. This design verifies or falsifies hypotheses and provides a cause and effect relationship between variables. For instance, in the aforementioned project it can help in determining the relationship

between dependent variables such as time (as a measure of performance), number of logic elements, input size and independent variables such as processor frequency (which is assumed to be constant).

3.4 Data Collection

Following the quantitative model, this project uses experiments for data collection [11] as a data set needs to be collected for a set of defined variables. For example, here, performance metrics are collected for a modified multiplier with defined input and output size using the Quartus Prime Timing and Performance Analyser. Other data collection tools such as questionnaires and case studies are mostly for the qualitative style and do not apply to the research at hand.

3.5 Data Analysis

The data collected using experiments as stated above is mostly analysed using computation [11]. Simulation is employed extensively, with an emphasis on algorithm use. In this project, the algorithm for a basic multiplier is made to incorporate Vedic mathematics and the resulting circuit is simulated and then the computed performance data is analysed to draw conclusions about the significant of this project. Furthermore, statistical data analysis [11] is also done to evaluate the significance of the results by comparing them to existing alternatives such as booth and array multipliers. Analysis methods such as coding, analytic induction and grounded theory, and narrative analysis require to be applied on qualitative data and hence cannot be used.

4 Results, Analysis and Discussion

We used Altera Quartus II development tool and Quartus Simulator along with ModelSim for our design. Our target device is Quartus Cyclone II EP2C50F672C8 FPGA which is kept uniform to analyse multiple multiplier architecture and their performances. Different multiplier logic units were coded using VHDL and the performance analysis is collected using simulators. TimeQuest Timing Analyzer tool was used to collect maximum frequency value, slack and hold times of the circuit. The maximum power dissipation was collected using PowerPlay Power Analyzer tool.

Booths multiplier

We have implemented an 8-bit booths multiplier in VHDL and simulated the result and performance using Altera Quartus II (Quartus II simulator). The booths VHDL component accepted two binary input, each of 8-bit length and produced 16-bit multiplication output with an average time delay of 16.38 ns as shown in Table I.

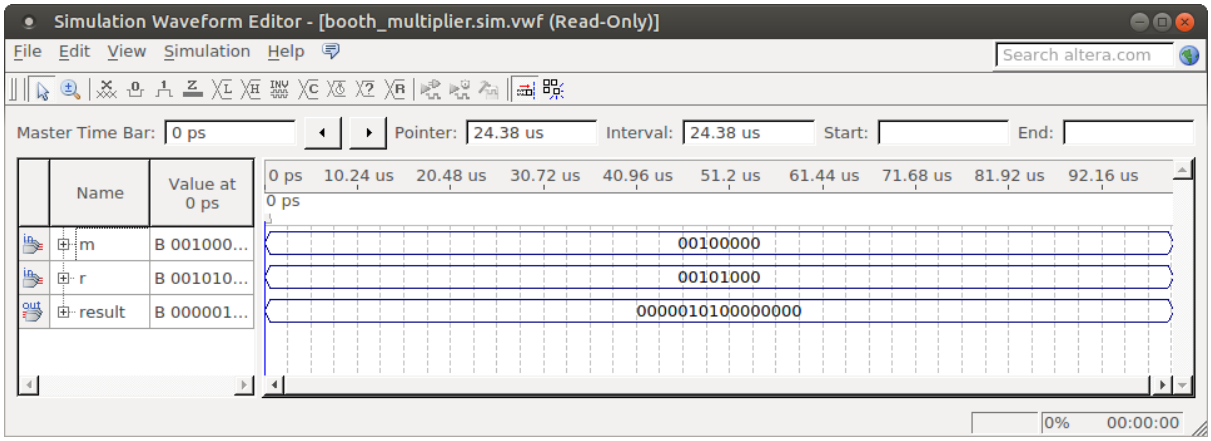

Figure IV: Simulated output of booths multiplication

Figure IV shows the simulation result for two positive binary inputs m (00100000_2), which is same as 32_{10} and r (00101000_2), which is same as 40_{10}. The result of the final product is stored in port 'result'(0000010100000000_2) whose decimal equivalent is 1280_{10}.

The booths multiplier device utilization is shown in the Table I.

Design Parameters	Actual Usage
Total logic elements	283 out of 50,528
Clock (MHz), F_{max}	103.7 MHz
Power Dissipation (mW)	197.08
Minimum period (ns)	16.38

Table I: Device utilization summary for booths multiplier

Vedic multiplier

To achieve maximum performance, we have implemented Vedic multipliers from bottom-up. i.e. first 2-bit multiplier is implemented and tested for correctness. 4-bit multiplier uses the already implemented 2-bit logic block in their code along with 4-bit adders to produce 8-bit multiplication output. 8-bit multiplier further reuses the 4-bit logic components in their design to achieve the final 16-bit multiplication result [5].

2-bit Vedic multiplier

The 2-bit multiplier implemented here uses the crosswise product and vertical product concepts of Vedic multiplication. Basic building blocks like AND gates and 1-bit adders are used to implement the multiplier logic as shown in Figure VI.

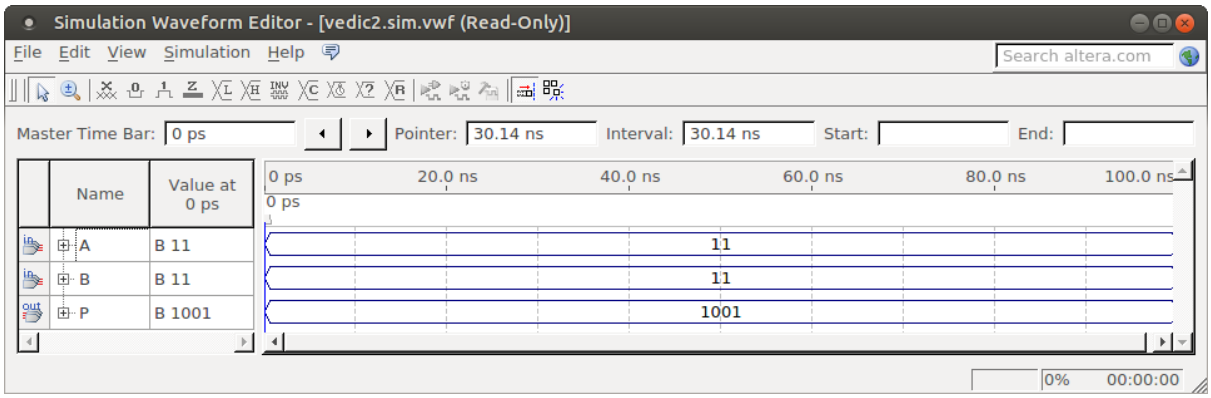

Figure V: Simulated output of Vedic 2-bit multiplication

Figure V shows the simulation result for two positive binary inputs A (11) and B (11). The result of the final product is stored into P (1001).

The 2-bit variant of Vedic multiplier device utilization is shown in the Table II.

Design Parameters	Actual Usage
Total logic elements	8 out of 50,528
Clock (MHz), F_{max}	340.02 MHz
Power Dissipation (mW)	147.00
Minimum period (ns)	3.7

Table II: Device utilization summary for Vedic 2-bit multiplier

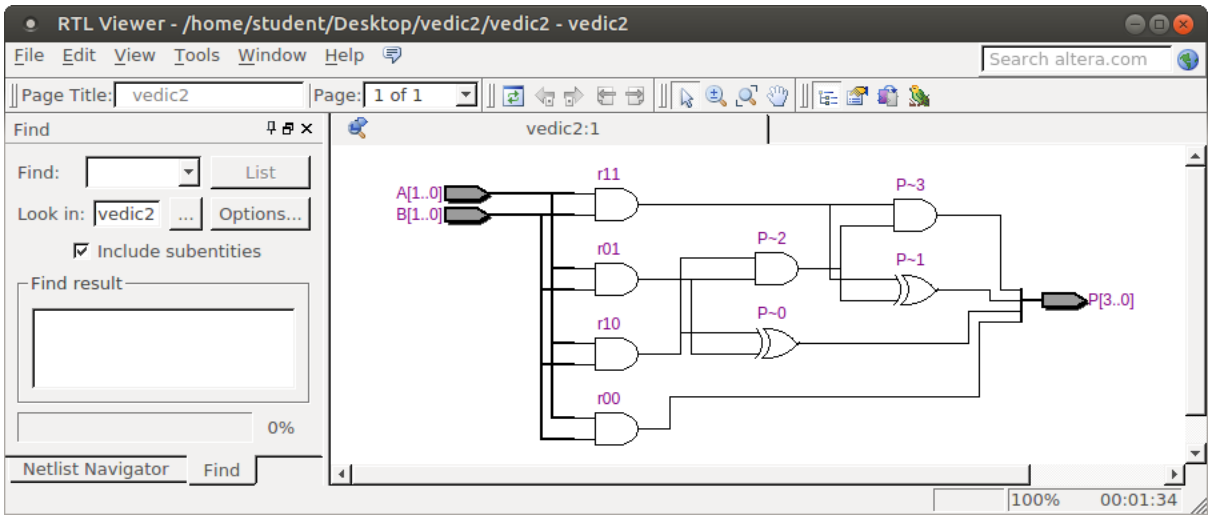

Figure VI: RTL Schematics of Vedic 2-bit multiplier

The actual netlist mapping of the 2-bit Vedic multiplier from Quartus II Simulator generated is shown in Figure VII.

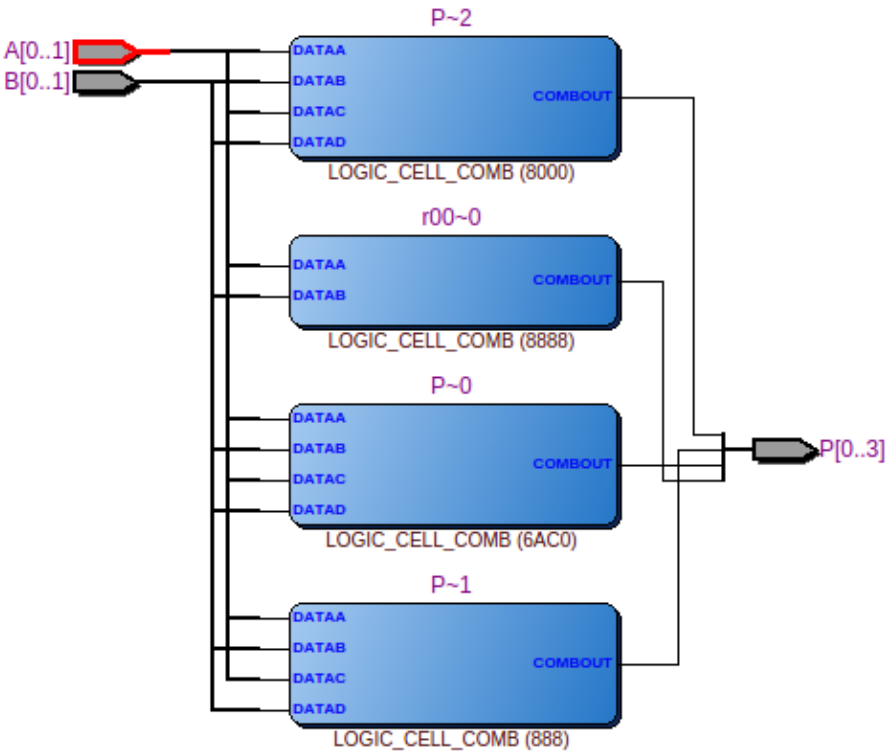

Figure VII: Post Mapping Netlist of Vedic 2-bit multiplier

4-bit Vedic multiplier

We instantiate four 2-bit Vedic multiplier block and three 4-bit full adder logic blocks to design 4-bit Vedic multiplier unit as shown in Figure IX.

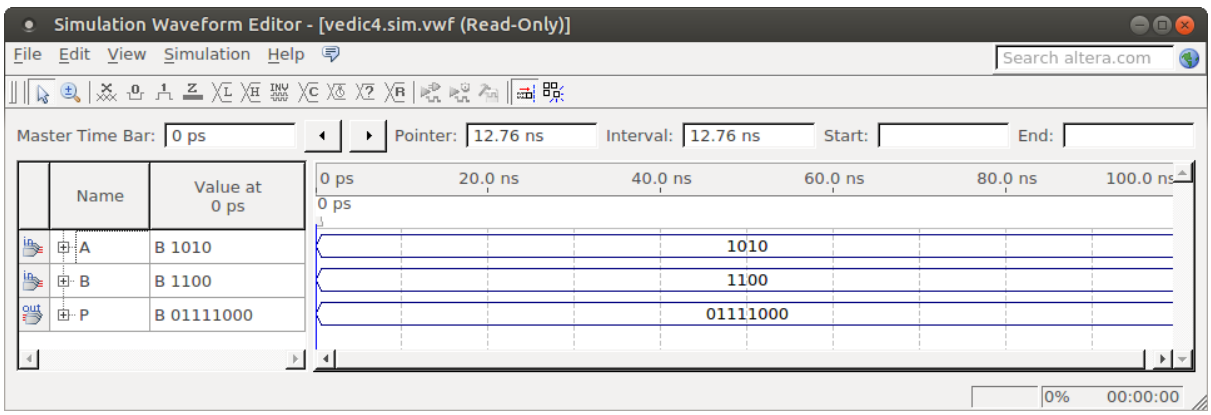

Figure VIII: Simulated output of Vedic 4-bit multiplication

Figure VIII shows the simulation result for two positive binary inputs A (1010) and B (1100). The result of the final product is stored into P (01111000).

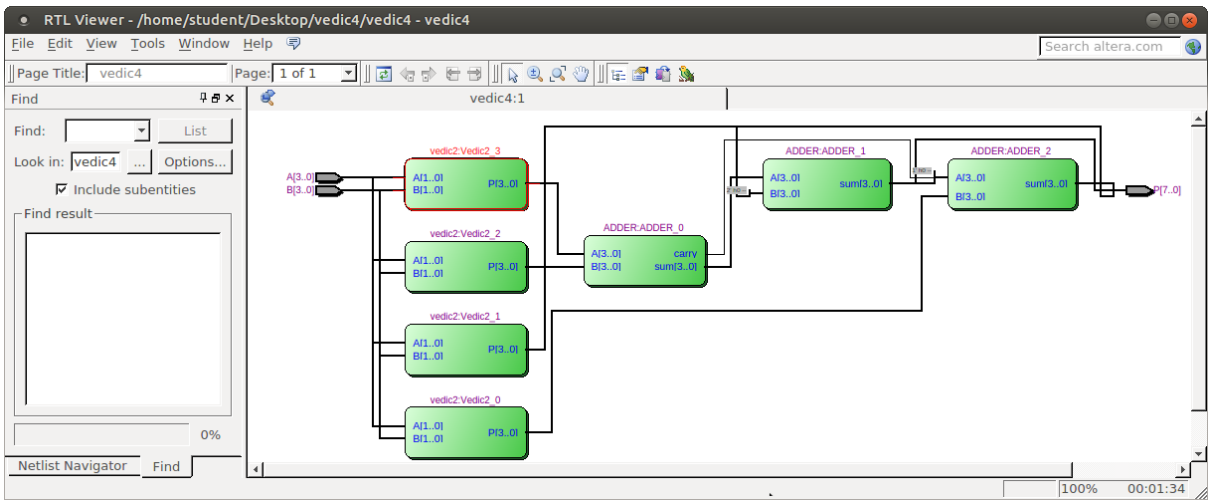

Figure IX: RTL Schematics of Vedic 4-bit multiplier

The 4-bit variant of Vedic multiplier device utilization is shown in the Table III.

Design Parameters	Actual Usage
Total logic elements	52 out of 50,528
Clock (MHz), F_{max}	250.19 MHz
Power Dissipation (mW)	147.96
Minimum period (ns)	9

Table III: Device utilization summary for Vedic 4-bit multiplier

8-bit Vedic multiplier

We instantiate four 4-bit Vedic multiplier block and three 8-bit full adder logic blocks to design 8-bit Vedic multiplier unit as shown in Figure XI.

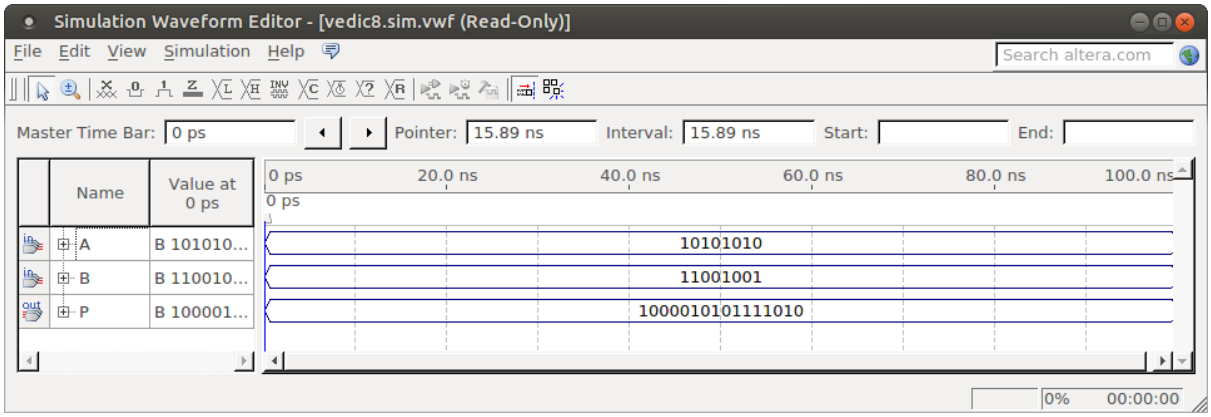

Figure X: Simulated output of Vedic 8-bit multiplication

Figure X shows the simulation result for two positive binary inputs A (10101010) and B (11001001). The result of the final product is stored into P (1000010101111010).

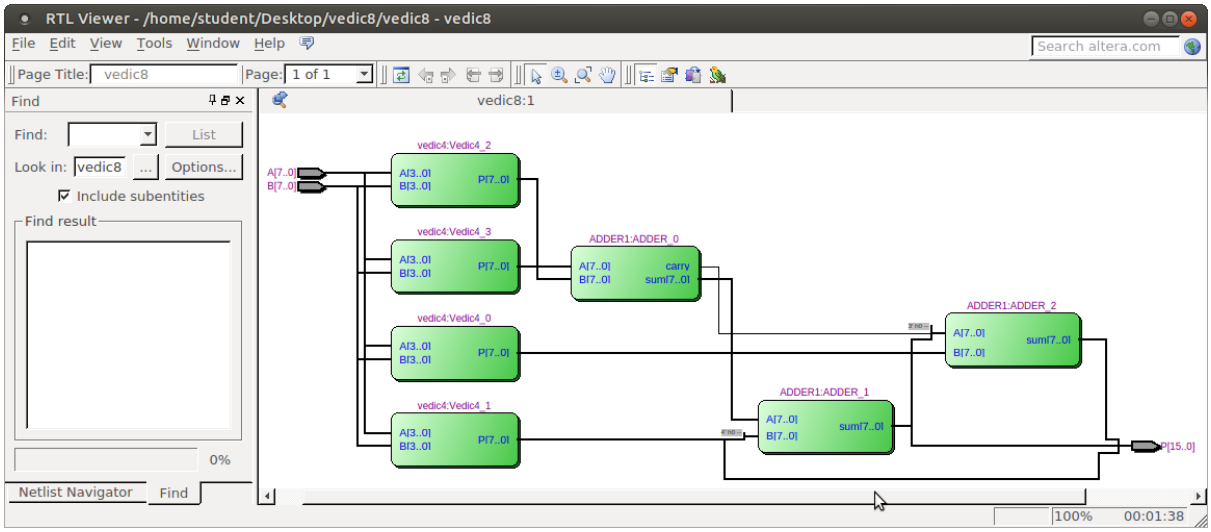

Figure XI: RTL Schematics of Vedic 8-bit multiplier

The 8-bit variant of Vedic multiplier device utilization is shown in the Table IV.

Design Parameters	Actual Usage
Total logic elements	141 out of 50,528
Clock (MHz), F_{max}	158.73 MHz

Power Dissipation (mW)	198.32
Minimum period (ns)	13

Table IV: Device utilization summary for Vedic 8-bit multiplier

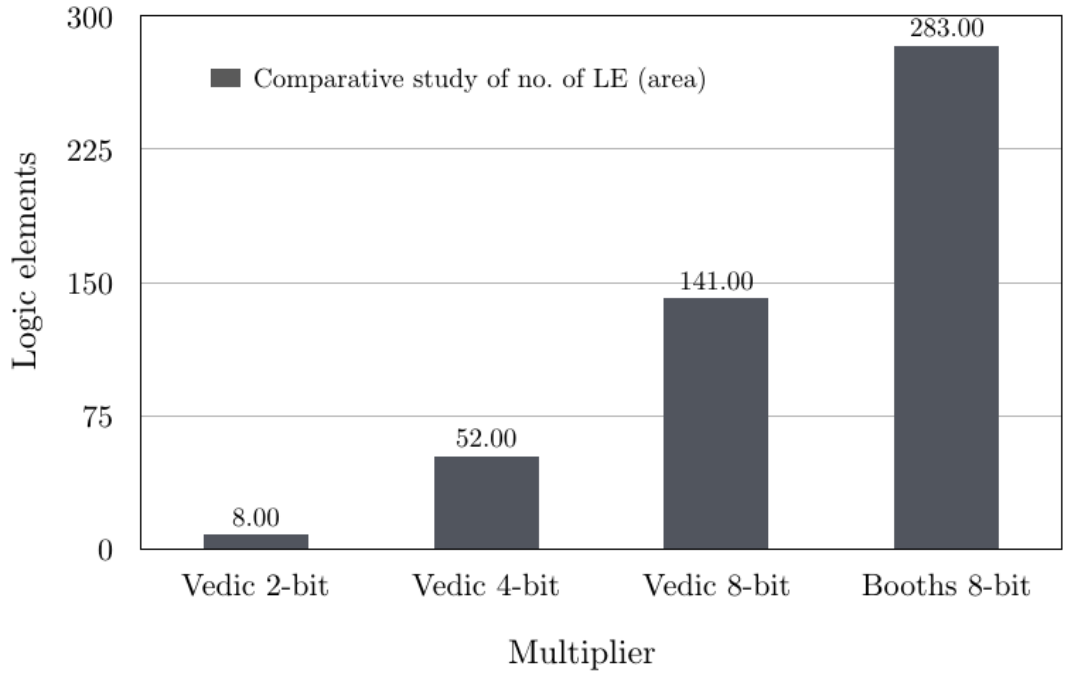

Figure XII: Comparison of number of Logic Elements used

Figure XII shows the comparative space taken up by different multipliers. Its clear from the graph that booths 8-bit take almost twice the number of logic elements as compared to Vedic 8-bit. Also the maximum allowed frequency with which we'll get a correct output (F_{max}) is observed to be higher in Vedic 8-bit case compared to the booths 8-bit counterpart in Figure XIII.

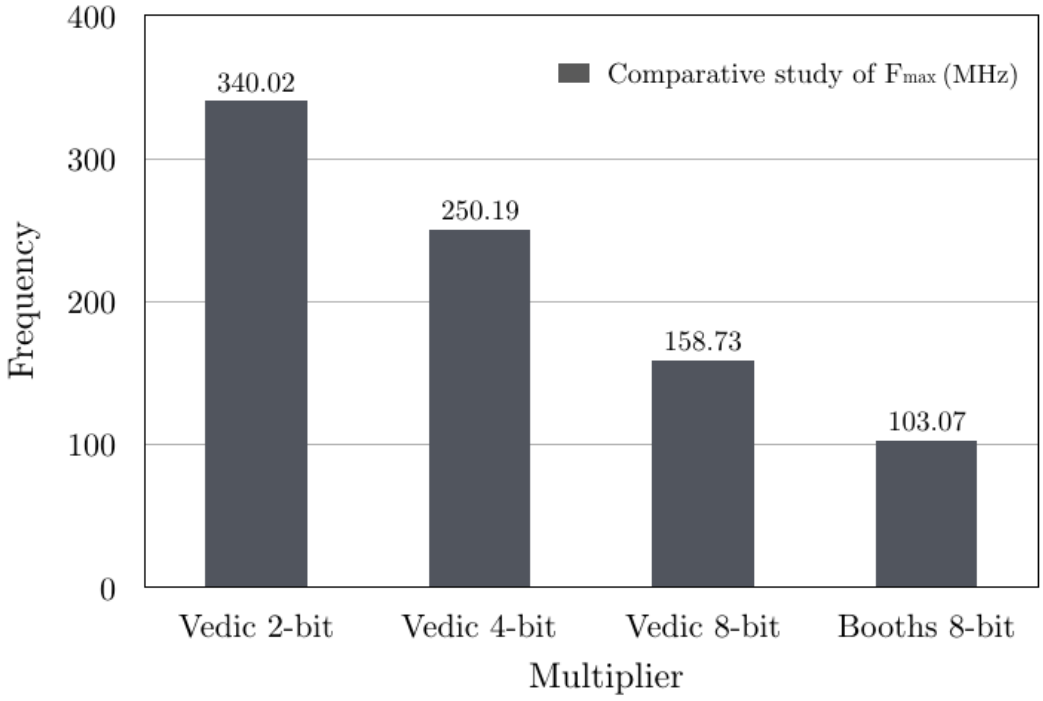

Figure XIII: Comparison of maximum frequency

5 Conclusion and future work

This project shows how Vedic multiplication can have a pronounced effect on the performance of a multiplier, especially with regard to area, power dissipation and execution time. There are several applications where a high-performance multiplier could elevate an entire system to an extent where the speed of the multiplier plays a major role in determining system performance. One particular application of interest is Montgomery Multiplication.

Calculating the required inverse employs a minimum of one multiplication, but usually more. The conversion of integers to corresponding Montgomery form involves one multiplication each mentioned in the description of [8]. Multiplying the Montgomery forms itself performs another (maximum of [log2(N) + 1] - bit) multiplication [8]. Each of the reductions performed also requires one multiplication each as mentioned in the REDC Algorithm in [8], thus making it one multiplication operation if the expected result is to be in the Montgomery form, or two operations if the expected result is to be in standard modular form.

From this, it is evident that a simple Montgomery multiplication operation uses a minimum of 6 multiplications in the best case. An average case scenario is arguably twice this number. While they are performed on relatively smaller numbers, the collective impact of these operations cannot be disregarded, especially on FPGA's

where multiplications usually tend to be extremely costly. Overall, a rough analysis points to the fact that close to 30% of system processing power is used for multiplication. There are indications that using the high-performance multiplier would result in a compounded improvement of at least 50% in multiplier area, a small amount in power dissipation and execution time.

References

[1] N. Nedjah, L. M. Mourelle, R. Reis, N. Calazans. Two hardware implementations for the Montgomery multiplication: sequential vs. parallel. Proceedings of the 15th. Symposium on Integrated Circuits and Systems Design, pp. 3-8, 2002.

[2] A. F. Tenca and C. K. Koc. A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm. IEEE Transactions on Computers, Vol 52, No. 9, pp. 1215-1221, 2003.

[3] M. Ghosh. Design and Implementation of Different Multipliers Using VHDL. Department of Electronics and Communication Engineering National Institute of Technology Rourkela, 2007.

[4] A.D. Booth. A Signed Binary Multiplication Technique. J. Mech. Appl. Math Vol 4, Part 2,1951.

[5] S. V. Mogre, D. G. Bhalke. Implementation of High Speed Matrix Multiplier using Vedic Mathematics on FPGA. International Conference on Computing Communication Control and Automation, pp. 959-963, 2015.

[6] Jagadguru Swami Sri Bharath, Krsna Tirathji. Vedic Mathematics or Sixteen Simple Sutras From The Vedas. Motilal Banarsidas, Varanasi India,1986.

[7] Vishal Verma and Himanshu Thapliyal. High Speed Efficient N X N Bit Multiplier Based On Ancient Indian Vedic Mathematics. In the Proceedings of the 2003 International Conference On VLSI(VLSI03),pp 361-365, Las Vegas ,Nevada, June 2003.

[8] P. Montgomery. Modular Multiplication without Trial Division. Math. of Computation, vol. 44, pp. 519-521, 1985.

[9] T. Blum and C. Paar. High-radix Montgomery multiplication on reconfigurable hardware. IEEE Trans. Computers, vol. 50, no. 7, pp. 759-764, July 2001.

[10] S. Eldridge and C. Walter .Hardware implementation of Montgomery's modular multiplication algorithm. IEEE Trans. Computers, vol. 42, no. 6, pp. 693-699, June 1993.

[11] A. Håkansson. Portal of research methods and methodologies for research projects and degree projects. p. 7.

[12] How to write a research methodology [Online]. Available: https://penmypaper.com/blog/how-to-write-a-flawless-research-methodology/.

[13] T. Blum, C. Paar. Montgomery Modular Exponentiation on Reconfigurable Hardware. Proc. 14th Symp. Computer Arithmetic, pp. 70-77, 1999.