LOW LATENCY MONTGOMERY MULTIPLIER FOR CRYPTOGRAPHIC APPLICATIONS

MUHAMMAD HUZAIFA

01-242181-003

A thesis submitted in fulfilment of the

requirements for the award of the degree

of Master of Science (Computer Engineering)

Department of Computer Engineering

BAHRIA UNIVERSITY ISLAMABAD

APRIL 2020

THESIS COMPLETION CERTIFICATE

Student's Name: Muhammad Huzaifa Registration No. 56187 Program of Study: MS-CE Thesis Title: LOW LATENCY MONTGOMERY MULTIPLIER FOR CRYPTOGRAPHIC APPLICATIONS.

It is to certify that the above student's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for Evaluation. I have also conducted plagiarism test of this thesis using HEC prescribed software and found similarity index at 11% that is within the permissible limit set by the HEC for the MS/MPhil degree thesis. I have also found the thesis in a format recognized by the BU for the MS/MPhil thesis.

Principal Supervisor's Signature: _________________________

Date: _________________________ **Name: Dr. Khalid Javed**

APPROVAL FOR EXAMINATION

Scholar's Name: MUHAMMAD HUZAIFA Registration No. 56187 Program of Study: MS(CE) Thesis Title: LOW LATENCY MONTGOMERY MULTIPLIER FOR CRYPTOGRAPHIC APPLICATIONS.

It is to certify that the above scholar's thesis has been completed to my satisfaction and, to my belief, its standard is appropriate for submission for examination. I have also conducted plagiarism test of this thesis using HEC prescribed software and found similarity index 11% that is within the permissible limit set by the HEC for the MS degree thesis. I have also found the thesis in a format recognized by the BU for the MS thesis.

Principal Supervisor's Signature ______________________________

Date: ______________________

Name: ______________________

AUTHOR'S DECLARATION

I, **MUHMMAD HUZAIFA** hereby state that my MS thesis titled **"Low Latency Montgomery Multiplier for Cryptographic Applications"** is my own work and has not been submitted previously by me for taking any degree from this university or anywhere else in the country/world. At any time if my statement is found to be incorrect even after my graduation, the University has the right to withdraw/cancel my MS degree.

Author's Name: Muhammad Huzaifa

Date: ___________________________

PLAGIARISM UNDERTAKING

I, solemnly declare that research work presented in the thesis titled **"Low Latency Montgomery Multiplier for Cryptographic Applications"** is solely my research work with no significant contribution from any other person. Small contribution / help wherever taken has been duly acknowledged and that complete thesis has been written by me.

I understand the zero tolerance policy of the HEC and Bahria University towards plagiarism. Therefore, I as an Author of the above titled thesis declare that no portion of my thesis has been plagiarized and any material used as reference is properly referred / cited.

I undertake that if I found guilty of any formal plagiarism in the above titled thesis even after award of MS degree, the university reserves the right to withdraw / revoke my MS degree.

Author's Sign: ___________________

Author's Name: **Muhammad Huzaifa**

DEDICATION

I dedicate thesis to the most merciful Allah, Everlasting praise to Allah the most Gracious who bestowed me with His great blessings. He who says in Quran:

"Indeed, I am near, I respond to the invocation of supplicant when someone calls upon me." (Surah AL-Baqarah; Ayat: 186)

I also dedicate this thesis to my family, teachers, and friends who carried out kind-heartedness, devotion and boundless support with me in this whole duration.

My teachers supported me with invaluable help of constructive remarks, suggestions, great opinions and inspirational thoughts during this journey which made me complete this research and they guided me to put trust in Allah, have faith in hard work and that so much could be ended with little. My intense gratitude is for them.

Muhammad Huzaifa

ACKNOWLEDGEMENTS

First and foremost, I am thankful to Almighty Allah for giving me opportunity, courage and vitality towards my ambition. His endless grace blessings and mercy enabled me throughout this journey of research, Emphatically, I cannot do anything without his unlimited succor and blessings. Whosoever helped me throughout the whole duration of my thesis, whether my parents or any other individual was his will, so assuredly none be worthy of praise but you.

My beloved parents, who raised me, facilitate me and courage me towards higher studies all my wholehearted thanks is for them.

I would also like to express my gratitude to my Supervisor **Dr. Khaild Javed** for his incredible cooperation and providing help at every phase of this thesis. He has guided me, encouraged me towards this thesis and his contribution has a major impact. Thank you for guiding me, often with big doses of patience and politeness.

ABSTRACT

In this modern era, protection of data is very important, to overcome these circumstances we can deploy different types of cryptographic algorithm such as VPNs, SSL and IPsec for securing our data from malicious attacks in many communication systems. Public channels are accessible to everyone, which is not safe for data and it cause high security risk. By having public-key cryptography, which provided secure communication between sender and receiver without need of sharing key at the beginning of communication. Public-key cryptographic systems such as ECC and RSA are implemented for different security services such as key exchange between sender, receiver and key distribution between different networks nodes and authentication protocols. Public Key (PK) cryptography is based on computationally intensive finite field arithmetic operations. Rivest, Shamir, Adelman (RSA) and Elliptic Curve Cryptography (ECC) are widely adopted public-key schemes. In these schemes, modular multiplication (MM) is the most critical operation. Usually, this operation is performed by integer multiplication (IM) followed by a Reduction Modulo M. However, the reduction step involves a long division operation that is expensive in terms of area, time and resources. Montgomery multiplication algorithm facilitates faster modular multiplication operation without the division operation. In this thesis, low latency hardware implementation of the Montgomery multiplier is proposed. Many interesting and novel optimization strategies are adopted in the proposed design. The proposed Montgomery multiplier is based on school-book multiplier, Karatsuba algorithm and fast adder's techniques. The Karatsuba algorithm (KA) and School-book multiplier recommends cutting down the operands into smaller chunks while adders facilitate fast addition for large size operands. The proposed design is simulated, synthesized and implemented using Xilinx ISE Design Suite by targeting different Xilinx FPGA devices for different bit sizes (64-1024). The proposed design is evaluated on the basis of computational time, area consumption, and throughput. It outperforms the state of the art.

TABLE OF CONTENTS

LIST OF FIGURES

List of Tables

ABBREVIATIONS

PK	-	Public Key
RSA	-	Rivest Shamir Adelman
ECC	-	Elliptic Curve Cryptography
NIST	-	National Institute of Standards and Technology
M	-	Modulo
ENIAC	-	Electronic Numerical Integrator and Computer
IC	-	Integrated Circuit
ASIC	-	Application-specific Integrated circuits
PCB	-	Printed Board Circuit
SPLDs	-	Simple programmable logic devices
PAL	-	Programmable array logic
PLA	-	Programmable logic array
GAL	-	Generic array logic
CPLDs	-	Complex programmable logic devices
FPGA	-	Field-Programmable Gate Arrays
FPIC	-	Field-Programmable Interconnect
VHDL	-	VHSIC Hardware Description Language

CMOS	-	Complementary Metal Oxide Semiconductor
DES	-	Data Encryption Standard
AES	-	Advanced Encryption Standard
DH	-	Diffie-Hellman
MD5	-	Message Digest algorithm 5
RC4	-	Rivest Cipher 4
DSA	-	Digital Signature Algorithm
MM	-	Modular Multiplication
MMM	-	Montgomery modular multiplication
CSA	-	Carry save Adder
DSP	-	Digital Signal Processor
BNC	-	Barreto-Naehrig curves
LUT	-	Lookup Table
KA	-	Karatsuba-Ofman Algorithm
IOB	-	Input Output Bounds
SB	-	School Book
CLB	-	Configurable Logic Blocks
IM	-	Integer Multiplier
FA	-	Full Addition
LR	-	Load Register

PPA	-	Partial Product Addition
PPM	-	Partial Product Multiplication
L	-	Load Register
W	-	Write to Register
A	-	Addition
C	-	Compression
S	-	Subtraction

CHAPTER 1.

Introduction

This section provides a detail introduction about the research and research concepts. This section is organized in multiple sub sections. **Section 1.1** provides the background study, **Section 1.2** presents the problem statement of research, **Section 1.3** discuss the thesis objectives, **Section 1.4** gives the detail about research contribution, and thesis organization is presented in **Section 1.5**.

1.1. Background Study

The purpose of this section is to introduce the background study of multiple concepts, which has been used in this research. These concepts include:

- Digital System Design.
- Network Security.
- Public Key Cryptographic.
- Elliptic Curve Cryptography (ECC)

1.1.1. Digital System Design

Transistors were utilized at first as discrete segments, however with the appearance of Integrated Circuit (IC) innovation, their utility expanded exponentially. ICs are reasonable when delivered in enormous numbers, dependable, and devour comparatively less force than vacuum tubes. IC innovation makes it conceivable to fabricate total computerized incorporating obstructs with single, minute silicon "chips". The size of transistors has been contracting since they get introduced to the world, and till today, a total PC is on one chip (microchip), and even huge systems are being coordinated into a solitary chip (system-on-a-chip). Chips designed to meet the particular prerequisites of an application are known as Application-Specific Integrated Circuits (ASICs) or specially crafted chips. The rationale chip is designed without any preparation. The rationale hardware is designed by the particulars and afterward executed in a proper innovation. The fundamental bit of leeway of ASICs are streamlined for a particular application, they perform superior to do practically comparable circuits worked from off-the-rack ICs or programmable rationale gadgets. They involve next to no zone, as the entirety of the rationale can be incorporated with one chip. In this manner, less PCB zone would be expected, prompting some cost reserve funds. The disservice of ASICs is that they can be legitimate monetarily just when there is mass creation of ICs. Normally, countless ASICs must be made to recoup the uses which are vital in the design i.e. assembling and testing stages. Another disadvantage of the special craft approach is that it requires crafted by exceptionally talented architects in the design, assembling, and test stages. The design time required for these chips is additionally high, a great deal of confirmation must be done to check for right usefulness. The hardware in the chip can't be changed once it is manufactured.

Now a days, DIGITAL SYSTEM DESIGN become increasingly famous in the field of Signal preparing (either in sound, pictures or other type of information) processing, correspondence, information stockpiling and numerous different fields thoroughly depend on system design. Emphatically, digital system become obligation of the computerized world which can't hold up under a solitary. Practically all hardware system is based after acquiring complete and careful guidelines. Obviously, certifiable signs are generally simple and interfacing to the outside world requires transformation of a sign (data) to computerized form. After attaining, effortlessness, adaptability, repeatability, and the capacity to deliver enormous and unpredictable (undoubtedly) system which financially make them tremendous for handling and putting away data (information). Summarizing that we conclude no one can envision without advanced system in the cutting-edge period of 21st century.

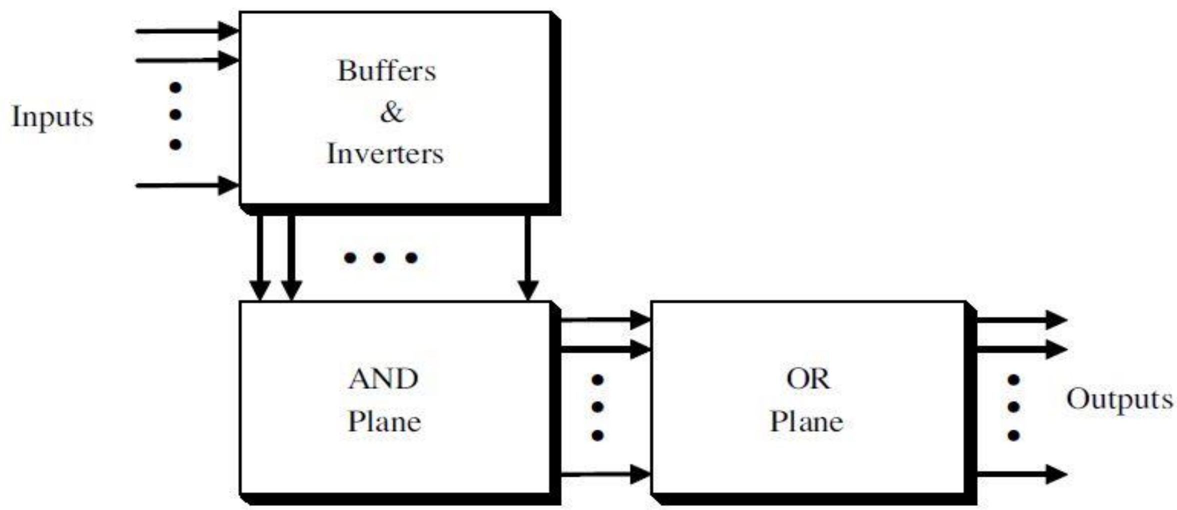

Figure 1-1: Basic Concept of SPLDs

Simple Programming logic Devices (SPLDs) assimilate Programmable Logic Arrays (PLAs) and Programmable Array Logics (PALs). Early SPLDs were basic and comprised of a variety of AND entryways driving a variety of OR gates. An AND gates (known as an AND plane or AND exhibit) sustains a lot of OR entryways (an OR plane). This aides in understanding a capacity in the entirety of-items structure. Following figure 1-1 shows rudiments of basic programmable rationale gadgets (SPLDs).

These chips have basic structure which ensure better understanding of programmable switches which permits a client to arrange it effectively to perform various capacities. The end client (developer) just change the arrangement of these switches. By composing a basic program in Hardware Description Language such VHDL or Verilog and consume it to chip. Most sorts of PLDs are reprogrammable for a fixed number of times (by and large, a high number). This component makes PLDs to turn out to be increasingly famous among the client, which can utilize it commonly by resetting it. Later it can effectively use in prototyping of standard chip. A designer can program a PLD to play out a specific capacity and afterward make changes and reinvent it for retesting on a similar chip.

The most acclaimed sort of PLDs include:

- Simple programmable logic devices (SPLDs).
- Programmable array logic (PAL).
- Programmable logic array (PLA).
- Generic array logic (GAL).
- Complex programmable logic devices (CPLDs).
- Field-programmable gate arrays (FPGA).
- Field-programmable interconnect (FPIC).

The same number of these chips having reprogrammable efficacy, so it is efficient like cost sparing by reuse philosophy which is a peculiar component that contributes in prototyping. A few focal points of these systems are recorded underneath.

- Ease of Programmability.
- Reduction in cost of Hardware.
- High Speed.
- High Reliability.
- Easy Design.
- Result reproduction is easy.

The fundamental impediment of PLDs is that they may not be the best performing. The presentation of a practically proportional ASIC or standard chip is probably going to be better. This is on the grounds that all capacities must be acknowledged from existing squares of rationale inside the PLD. Various kind of PLDs have diverse inward circuit which allude various disservices.

1.1.2. Network Security

System security comprises of approaches and practices which are actualized to forestall and follow illegitimate access, abuse, adjustment or dismissal of a PC arrange and accessible system assets. Security is a hazard evaluation. Secure condition doesn't show up, they can be designed and created through exertion. Secure arrangements must be finished in all angles, one shortcoming may concord the entire secure system.

Presently, increasingly associated by means of Internet, PCs, TV's, and web-based business deal, and all other online world secure system condition are consistently developing. Shortcomings inside the system have prompted the fast development of wholesale fraud and day by day PC infection flare-ups. Programming engineers and system chairmen are being considered responsible for concords inside their systems, while aggressors stay unknown.

Inside the security network, there conquer explicit implications, though different words normally connected with PC security.

Vulnerability: A deformity or shortcoming in the possibility, design, usage, activity, or upkeep of a system.

Threat: A foe who is able and aroused to misuse a vulnerability.

Attack: The utilization or misuse of a vulnerability. This term is neither pernicious nor considerate. A trouble maker may assault a system, and a hero may tackle an issue.

Attacker: The individual or procedure that starts an assault. This can be synonymous with danger.

Exploit: exploit tends better and effective use of resources. Resources that can be utilized for an assault. A solitary weakness may prompt various adventures, yet few out of every odd defencelessness may have an endeavour (e.g., hypothetical vulnerabilities).

Target: An individual, organization, or system that is forthright defenceless and affected by the adventure. A few endeavours have numerous effects, with both essential (fundamental) targets and auxiliary (accidental) targets.

Attack vector: The way from an assailant to an objective. This incorporates devices and procedures.

Defender: The individual or procedure that mitigates or forestalls an assault.

Compromise: The effective abuse of an objective by an assailant.

Risk: A subjective evaluation depicting the probability of an assailant/danger utilizing an endeavour to effectively sidestep a protector, assault a powerlessness, and bargain a system.

Characteristic mental attitude determines that security is utilized to keep out maddening components and keep fair individuals legit. In fact, security doesn't simply mean sparing the system from miscreants and noxious programming. Security implies saving information respectability, giving approved access, and looking after protection. Along these lines, keeping a client from inadvertently erasing or defiling a significant record is similarly as fundamental to security as halting a vindictive client. By far most of security issues are focused on information insurance and protection issues, guaranteeing that clients don’t accomplish something. There are numerous approaches to analyse, organize security issues. As a rule, a danger can either originate from a record on the system (neighbourhood get to), or from a system over a system (remote access). Be that as it may, somebody with physical access to a system may further represent a risk.

Security does not signify "safe." Even the most verified PC system will presumably lose information in the event that it is close to a solid electromagnetic heartbeat (i.e., atomic impact). Security implies that in a general-use condition, the system won't be straightforwardly defenceless against assaults, information misfortune, or protection issues. Assailants may even have the option to get verified by a system. However, it will be considerably harder for them, and assaults might identify effectively.

1.1.3. Public Key Cryptographic:

There are three fundamental ways to deal with verifying data i.e. counteraction, limitation, and cryptography. Access to data can be forestalled. In the event in which an aggressor doesn't have access to data, at that point the data is protected from the assailant. For organize security, separated systems and prohibitive models are commonly adequate hindrances Public-key cryptography, or deviated cryptography. They are cryptographic system that utilizations set of keys i.e. open keys which might be dispersed broadly. Private keys are known uniquely to the proprietor. The age of such keys relies upon cryptographic calculations. They dependent on scientific issues to deliver single direction capacities. Powerful security requires private key not to disclose; the open key can be straightforwardly conveyed without trading off security. Encryption and decoding are done utilizing two unique keys. The two keys in such a key pair are alluded to be the open key and the private key.

Cryptography is the most widely recognized methodology for verifying systems. Cryptography encodes information by having primarily objective of solitary which expect devisee can interpret the message. Cryptographic systems incorporate irregular number generators, hashes, figures, and encryption calculations. Concealing data is generally connected with cryptographic calculations, this phenomenon is said to be Steganography.

Each cryptographic procedure has five fundamental components i.e. the calculation, condition, key, plaintext, and cipher text. Encryption required all of these segments. However, an assailant may know at least one of these necessary components and use them to decide different components.

Cryptography is utilized to shield information from unintended beneficiaries. The decoded information is called plaintext, however the information may not really be content. Pictures and double records may further be scrambled. Scrambled plaintext is called cipher text. Cipher text gives secrecy by keeping unapproved beneficiaries from review the plaintext.

An encryption calculation is utilized to change over plaintext (P) into cipher text (C) and the other way around. This requires encryption (E) and decoding (D) capacities, with the end goal that

$$E (P) = C$$

$$D(C) = P$$

Every encryption makes cipher text that can be unscrambled into plaintext. Rehashed encryptions may produce diverse cipher text, yet the first plaintext can generally be recouped.

Cryptographic conditions indicate usage explicit alternatives. Encryption calculations (e.g., DES, Blowfish, and AES), hash capacities (e.g., RC4, MD5), and key trades (e.g., DH and ADH) are all around characterized however shift between stages. For instance, the Diffie-Hellman key trade (DH) can utilize enormous whole numbers. One usage may utilize 64-piece whole numbers, while another utilization 256-piece numbers. Albeit distinctive piece sizes do not affect the science behind the calculation, it impacts stage similarity. In the event that the earth is obscure, it tends to be as compelling at hindering an assailant. It got happen by utilizing a novel encryption system.

Cryptographic calculations convert plaintext to cipher text in a no predictable manner. The subsequent cipher text cannot be foreordained from the plaintext. Most cryptographic systems join irregular number calculations. The irregular number generator (R) is seeded with a particular worth. The seed and plaintext create the cipher text. A key (K) might be utilized as the seed esteem or joined with a hash capacity to create the underlying seed esteem.

$$E (K, P) = C$$

$$D (K, C) = P$$

Without a key, the calculation perform encoding, not an encryption. The equivalent plaintext will consistently produce the equivalent cipher text. Any assailant realizing the calculation can promptly decipher the cipher text. Encryption calculations use keys to change the cipher text; plaintext encoded with various keys creates distinctive cipher text. Without knowing the right key, an aggressor can't decipher the cipher text.

1.1.4. Elliptic Curve Cryptography (ECC)

An open key encryption method dependent on ecliptic curve hypothesis that can be utilized to make quicker, little and progressively productive cryptographic keys. ECC creates keys utilizing properties of the elliptic curve condition rather than the conventional strategy for age as the result of exceptionally enormous prime numbers. The ECC (Elliptic Curve Cryptography) calculation was initially freely recommended by Neal Koblitz (University of Washington), and Victor S. Mill operator (IBM) in 1985.

In spite of the fact that the ECC calculation was proposed for cryptography in 1985, it has had a moderate beginning and it took about twenty years, until 2004 and 2005, for the plan to increase wide acknowledgment. ECC (Elliptic Curve Cryptography) is a generally new calculation that makes encryption keys dependent on utilizing focuses on a curve to characterize people in general and private keys.

As more individuals moving towards cell phones, ECC key is useful for the present age. Perhaps, use of Smartphone stretches out to develop. There is rising requirement for a progressively adaptable encryption for business to meet with expanding security necessities. Beside this, we contrast with the RSA and DSA calculations, at that point 256-piece ECC is equivalent to 3072-piece RSA key. The purpose for keeping short key is the utilization of less computational force, quick and secure association, perfect for smartphone and tablet as well. The US government and the National Security Agency have guaranteed ECC encryption technique. The scientific issue of the ECC calculation is more diligently to break for programmer's contrast with RSA and DSA, which implies the ECC calculation guarantees site and framework wellbeing than customary strategies in a progressively secure way.

If we inspect Table 1, there is an extensive development in DSA and RSA key than ECC key size. A more drawn out key requires more space, more data transmission, and extra processor power. Emphatically, it will set aside an effort to create a key, encode information, and decode the information.

Table 1: NIST guidelines for Key Sizes

Date	Security Bit	(AES) Symmertic Algorithms	RSA	ECC	ECC:AES	RSA:ECC
2010	80	2-key triple DES	1024	160	2:1	6.4:1
2011-2030	112	3-key triple DES	2056	224	2:1	9.14:1
>2030	128	AES-128	3072	256	2:1	12:1
>>2030	192	AES-192	7680	384	2:1	20:1
>>>2030	256	AES-256	15360	512	2:1	30:1

Encryption specialists are squeezed to discover perpetually compelling strategies, estimated in security and execution, in light of the fact that dangers exhibited by programmers are ever more worthy. Mostly on the grounds that the programmers themselves become progressively complex in their assaults, and furthermore that the aftermath from an assault gets always hazardous as our utilization of information develops.

It makes an earnestness of new calculations with an objective to give more significant level of security by having keys that are increasingly hard to break, while offering better execution over the system and focusing that they are working with huge informational collections. A few variables are adding to its expanding notoriety. Most importantly, the security of 1024-piece encryption is debasing, because of quicker processing and a superior comprehension and examination of encryption strategies. While beast power is still improbable to split 1024-piece encryption. Different methodologies including exceptionally escalated equal figuring in disseminated registering clusters are bringing about increasingly advanced assaults. These assaults have diminished the adequacy of this degree of security. As, even 2048-piece encryption is evaluated by the RSA security to be compelling just until 2030.

Entrepreneur require information regarding web server models. Many web servers running on a solitary area name can deal with RSA, DSA, and ECC arrangement. Meanwhile, not many web servers are be able to deal with various calculations and can use a solitary endorsement on a solitary web server.

RSA, DSA, and ECC have assorted speed for confirmation and verification. RSA is a quick calculation as far as customer confirmation while ECC is quicker regarding server verification. Mark confirmation got quick if there come an occurrence of RSA key contrasting with ECC key. There are exchange types, the preparing intensity of the gadget; stockpiling limit, transmission capacity, and utilization of intensity additionally impact the calculation choice.

Numerous administration elements have begun to acknowledge DSA and ECC. They required for government subcontracts, government branches for their inward trade of correspondence.

The quantity of associations assumes an imperative job in choosing calculation standard. ECC can deal with more associations. Simultaneously more contrasts with RSA calculation can be assigned. An organization needs to keep up the harmony between security, client experience, and IT-framework cost engaged with arrange process.

1.2. Problem Statement

In every Public-key cryptosystem, the modular multiplication is the important operation. The most broadly utilized calculation for the execution of modular multiplication is the Montgomery modular multiplication architecture of school-book multiplier and Karatsuba algorithm with different splitting parts which enable us to utilize suite in dedicated FPGA multipliers. The problem statement of this thesis is described as, Modular multiplier effectuate as bottleneck in many public key cryptographic schemes. The modular multiplier's overall performance affects the performance of the cryptographic high-speed scheme. This work proposed dedicated hardware accelerated to compute modular multiplier operation for different bit sizes.

1.3. Thesis Objectives

Entire research is done in a very systematic way. **Figure 1-2** represent the stepwise flow of research. In first step we identify the problem. Then proposed the ideal solution for the problem which was presented earlier. We carry out a detailed and comprehensive literature review which helps us to identify the optimal solution for the problem. We reviewed the researches carried out related to our proposed solution, analyse and compared it.

The proposed solution includes the Hardware implementation of public key cryptographic algorithms. Montgomery modular multiplier pay significant role. This proposed methodology provide a full-word implementation of Montgomery modular

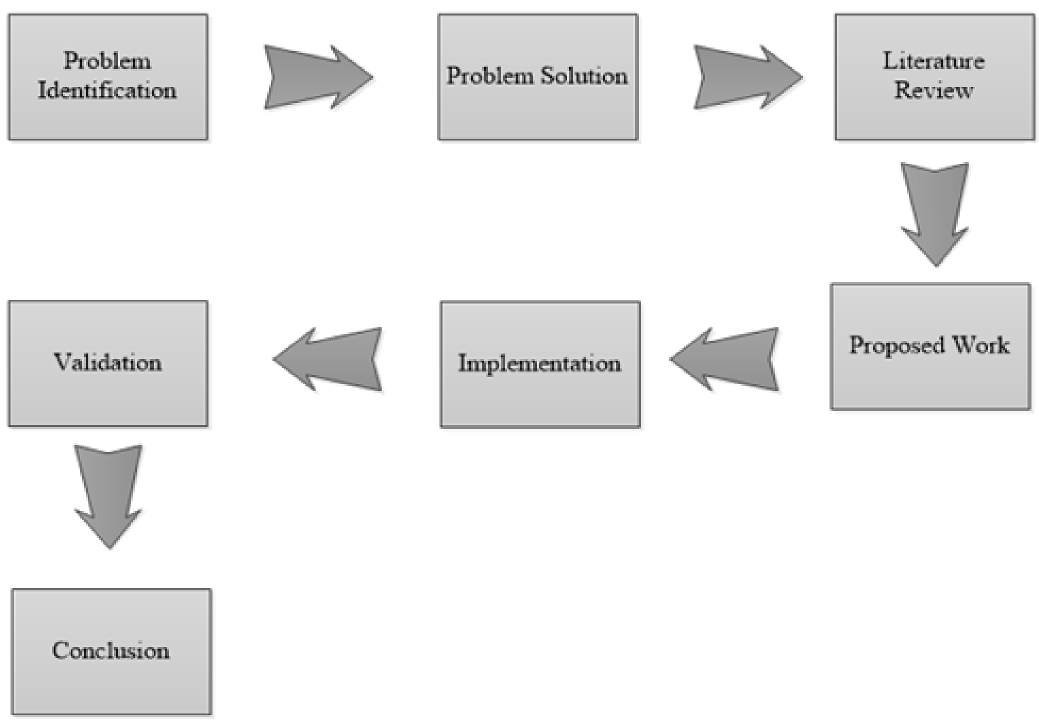

Figure 1-2: Research Flow

multiplication which enhance the execution speed of Elliptic-curve cryptography (ECC) and RSA cryptographic algorithms on hardware. We also utilized the school-book multiplier and Karatsuba–Ofman algorithm to calculate the 64-1024.

Multiplication using Xilinx FPGA devices. We optimized the Model using the school-book multiplier and Karatsuba–Ofman algorithm techniques to divide the operand on different size according to Xilinx FPGA devices and the adders enable fast addition by reducing long carry propagation delay. The proposed methodology has been validated by computational time, area consumption, and throughput. It significantly surpass the state of the art.

1.4. Research Contribution

Contributions in this research includes split the operands in different parts using school-book multiplier and Karatsuba-Ofman algorithm, apply the multiplication operation on the splitting part of the operands by using the applications of Montgomery modular multiplication. Detailed set of contributions of the proposed approach are as follows:

- We have utilized the model of Montgomery multiplication shift and add operations replacing the cost of the division operation.
- We have presented a model, which divide the operands into two, four and eight equivalent small parts to overcome the complexity.
- This model consists of a School-Book algorithm which can do fast multiplication operation and Karatsuba-Ofman algorithm which can save one multiplication operation.
- We have provided validation of our work with different splitting parts which enable us to utilize suite in dedicated FPGA multipliers.

1.5. Thesis Organization

Figure 1-3 represent the organization of thesis**. Chapter 1** prescribe introduction having detailed background study about the concepts used in the research, problem statement, research contribution and thesis organization. **Chapter 2** contains the literature review which provide a description of work done in the field of Montgomery multiplication using school-book multiplier and Karatsuba-Ofman algorithm. In Literature review, we also highlight the research gaps that we encountered. It covers the detail of proposed methodology used for identification of problem. **Chapter 3** presents the detailed implementation regarding the proposed model, Montgomery multiplication, school-book multiplier and Karatsuba-Ofman algorithm. It provides the validation performed for our proposed methodology by comparing with other Montgomery multiplication models. **Chapter 4** contains a brief analysis of our proposed work with previous researches. **Chapter 5** include a brief discussion on the work done, contains the limitations to our research, conclude the research, and recommends a future work for the research.

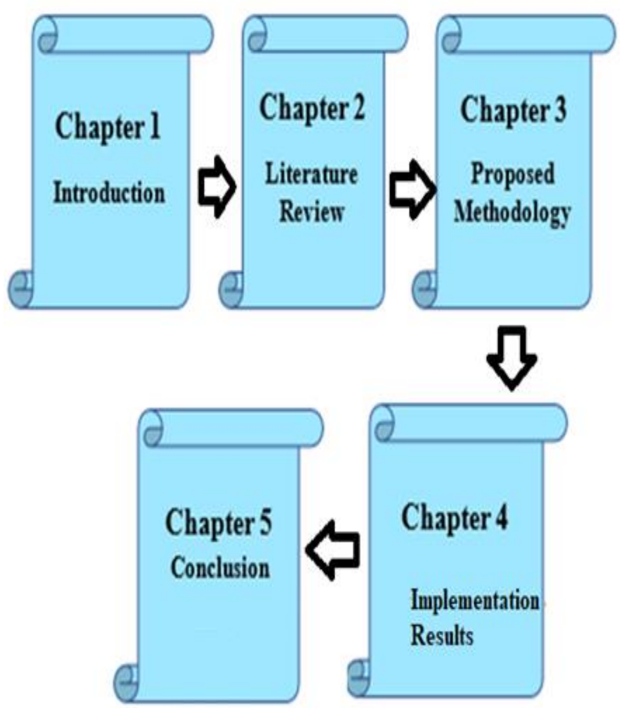

Figure 1-3: Thesis Outline

CHAPTER 2.

Literature Review

This chapter presents brief review of Montgomery Multiplier for Cryptographic Applications and existing work of efficient implementation of Montgomery modular Multiplier. After a brief literature review of work conducted in this area, we enlightened the research gaps that we found in previous works.

2.1. Literature Review

Cryptography has generally two type's i.e. symmetric cryptography and asymmetric cryptography [36]. In symmetric cryptography, encryption and decryption use the same key format but in asymmetric cryptography there is a pair of keys like public key and private key for encryption and decryption, it's called public key cryptography. The key represents itself, the public key is open accessible for everyone, but the private or personal key solely known by the message receiver. Both keys have mathematical relation in-between. In asymmetric cryptography, for the encryption of data public key is used while for decryption only private key is used. It is impossible to derivate the private key from the public key. It is allowed to freely exchange public key over the network.

Public key cryptographic algorithms are widely used in RSA and ECC [37]. Elliptic curves on finite fields are supported on the algebraic structures in the ECC. In public key cryptography, algorithms need arithmetic operations like modular multiplication, addition/subtraction and inversion/division functions with large size operands. Increased security schemes use large operands in operations. In RSA large operands are used for the same security level as compared to ECC. In public key cryptographic algorithms, the ECC is preferred scheme, particularly when affected resource environments are used.

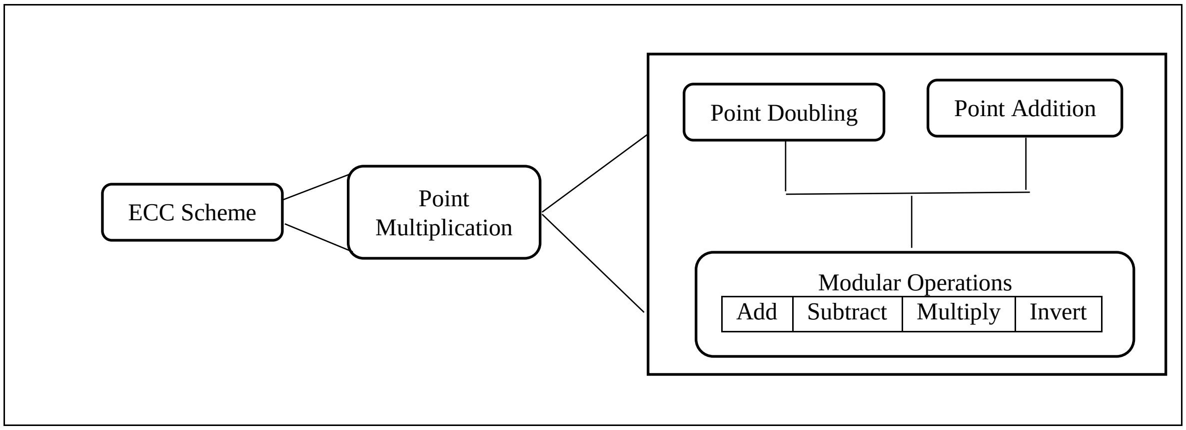

Figure 2-1: ECC Algorithm

The software implementation of ECC at the cost of time, occur complexities with a high level of flexibility [38]. The hardware implementation of ECC is done with high working speed due to effective utilization of modular multipliers. These layers of modular multipliers consist of mathematical operations like multiplication, addition, subtraction, and inversion.

The operations as modular multiplication and addition are mostly utilized as the hardware resources and cost of time. The modular multiplication is comparatively faster in terms of hardware and software in modular inversion function. Extra multiplications are added to remove the complexity of the modular inversion and the affected projective by differing the coordinate system. Adopting the projective coordinate system in ECC algorithm the total performance depends on the modular multiplication, which causes the bottleneck in cryptosystems. For increment in the performance of the ECC algorithm, we use the common methods.

Extensively, modular multipliers divided into three types. The regular technique for modular multiplication is division-using modulus M. In division the expensive operation in terms of execution and consumption of resources is Modulus M. The second type of modular multipliers is interleaved modular multiplication, which can do the reduction during multiplication. The last type of modular multiplier is Montgomery modular multiplication, which is faster for the large operands as compare to the other methods [41]. Large operand divided into a number of small parts and then use the school-book (SB) and Karatsuba-ofman algorithm (KA) [12] which give us best output in term of area and performance.

Cryptography applications deploy in several smart devices like mobiles and Wi-Fi devices. The Cryptography applications utilized the large size of mathematical operands starting from 160 to 2048 bits. Extended the security of the devices by increasing the size of the keys. For fast computing, it is required to speed up the computation and cut down the resources. With the public key crypto algorithms to scale down the computational cost in terms of space and delay to fulfil the requirement of portable devices. Cryptography applications uses the modular multiplication, which utilized the high space. Speeding up the cryptography applications is depending on space and time complexity. In modular multiplication operation, mostly use carry-save adder and high-radix multipliers, which create the carry propagation and caused the longest path delay and hardware complexity. Montgomery modular multiplication mostly uses the implementation of the RSA algorithm. An array of AND gates and carry-save adder are used to intermediate additions to implementation of the multiplier in hardware.

2.2. Related Work

As, concept of efficient implementation of Montgomery modular multiplier based on hardware. All the concept of implementation is divided into two major classes. Word wise implementations and bitwise implementations. In modern FPGA dedicated multipliers are not used in bitwise implementations, it uses only standard FPGA. The utilization of dedicated multipliers on FPGA is faster than the standard design-based FPGA. The word wise implementation divides the operands into different parts and these parts utilize the dedicated multipliers for multiplication, which offer faster speed in time-critical applications.

Dedicated multipliers are used for several implementations and for additional use of the fast carry adders. Mondal et al. in [1] a provided concept to use 64x64 bit cores architecture for different implementation and their resources regarding hardware architecture Brinci et al. in [2] presented concept for the multiplier of Barreto-Naehrig curves. They allotted the special prime number for implementations of BN curves and utilized the non-standard division to adjustment of the FPGA Digital Signal Processor (DSP) block. Kiang et al. in [3] extended concept to high speed & low-cost Montgomery multiplication algorithm with the Carry-Save Adder for added operations. Carry-Save Adder is used to cut off the extra clock cycles for implementation and conversion. Rezai et al. [4] advanced the concept of high-speed. Montgomery multiplier architecture using the digit serial computation. It uses the binary multiplication in high-radix partial multiplication. Consecutive zero-bit multiplications can be functioned within one clock cycle. The structural unit is using changed right-to-left modular operands architectures. K. Javeed [5] bestowed the concept with the addition of 512-bit and multiplication of 256-bit utilization of the carry chain of 64-bit with soft-core multiplier and succeed to 188 MHz frequency. Yang et al. in [6] elaborated concept of implementation scheme of using IP cores of the FPGA with the addition of 512-bit and 256-bit multiplication to achieve 50% better results as compared to standard implementations. C. J. McIvor [7] contributed to implement the hardware processor for ECC. Full block Montgomery modular multiplication with the 256-bit integer multiplier is intended of 16-bit cascading unsigned multipliers and this method is sustained until the specified size of the multiplier is achieved. Fast carry look-ahead adders are used for the addition of the modular multiplication. M. Morales-Sandoval [8] put up the design concept to divide the operands and performing computations on it. The complexity is not depending on the operand size, it relies on the divided part of the operand.

To increase the efficiency of Montgomery modular multiplication architecture many solutions are provided using the Karatsuba algorithm. Gong et al. in [9] contributed that the design concept of 256-bit Montgomery modular multiplication using the dedicated multiplier on FPGA with pipelining stages. Using the Karatsuba algorithm, the operands are divided into two parts to cut down the number of the multiplier on FPGA. The hardware architecture decreased the clock cycle of the output and providing limited frequency. S. Ghosh [10] put up the design concept of a series of nine multipliers of 64bits to create a block to deploy the

Karatsuba algorithm-based number multiplier. The number multiplier can be used to create a huge block of 256-bit Montgomery modular multiplier. The design provides low space architecture and path delay cost is decreased to increased iterations. G. C. Chow [11] presented the design concept when FPGA working of high-frequency, routing delay is increased in large multipliers. For decreasing the routing delay dividing the operands into small parts and these parts are using the dedicated multipliers to increase the efficiency of the hardware. I. San, N. At [12] extended the design concept of the Karatsuba-ofman algorithm (KA) to dividing the operand into two parts and use in the Montgomery modular multiplication algorithm with higher radix. Architecture uses the dedicated blocks of the multiplier, which increase the space of the hardware. X. Yan [13] advanced the design concept of the Karatsuba algorithm divided into 4 levels use in Montgomery modular multiplier. Using the splitting method operand is divided into two parts. The divided part again divided into two other parts in the reappearance style until the divided parts are length matches with the DSP blocks of an FPGA. Utilized the LUTs on FPGA rather than the dedicated multipliers. K. Javeed [14] enlightened the design concept of LUT use on the hardware implementation rather than the FPGA dedicated multipliers. Radix-4 based modular multiplier with the serial interleaved. K. Javeed [15] came up with the design concept of parallel interleaved modular multiplier implementation of hardware architecture. According to the architecture, an operand is working on four parallel processing elements to complete the dedicated task according to the algorithm. S. B. Ors [16] presented the design concept to deploy of systolic architecture array in Montgomery modular multiplier. Systolic architecture array repeating the structures in parallel to overcome the path delay. K. Javeed [17] provided the design concept to deploy a radix-4 serial multiplier in Montgomery modular multiplication with the laddering method of power to cut off the 50% in clock cycles.

Modular multiplication implementation presents four essential methods like multiplication and division [18], Brickell algorithm [19], Montgomery modular multiplication architecture [20], multiplication and minimization of interleaved [18]. Multiplication minimization of interleaved and Montgomery modular multiplication architecture has less hardware which is said to be computational technique [18]. The multiplication and minimization of interleaved, the n-bit number A and B are multiplied which results as the product is 2n-bits. Then the result is divided by the mod M to get the again result which is n-bit ((A x B) mod M). In the multiplication algorithm, there he utilized Shift and addition operation to minimize the result in every step [18]. Brickell algorithm is the utilization of the integer carry delay [19]. Sign estimation is the combination of this scheme and correction by Omura's algorithm [18]. Residue integer System [21], systolic architecture array [22], LUT [23] and pipeline architecture [24] are also utilized the hardware architecture optimization schemes. The minimization Interleaved Multiplication [25], [26] Montgomery modular multiplication architecture is used for the implementation of a binary base [27] FPGA architecture.

2.3. Research Gap

The proposed solution includes a cost-effective Montgomery multiplier design based on school-book multiplier, Karatsuba algorithm and adders. School-book multiplier and Karatsuba algorithm with different splitting parts enable us to utilize suite in dedicated FPGA multipliers. The adders enable fast addition by reducing long carry propagation delay. The design will be optimized for speed and hardware resource. The architecture will be synthesized and implement using Xilinx ISE 14.1 Design Suite for Virtex-5, Virtex-6, and Virtex-7. The architecture will be suitable for ECC and RSA implementation with different field sizes from 64-1024 bits. It gives the best outcomes as far as throughput and delay-area, so it tends to be proficiently utilizable in the Public key cryptography scheme field.

CHAPTER 3.

Proposed Methodology

In Public key crypto system, the modular multiplication (MM) is the basic operation. Montgomery modular multiplier (MMM) algorithm is most widely used for the implementation of modular multiplication. To increase the efficiency of Montgomery modular multiplier algorithm, employed the School-book multiplier and Karatsuba-Ofman algorithm. Reduce the complexity of multiplication utilize the School-book multiplier and Karatsuba-Ofman algorithm which divide the operands into smaller chunks. Before multiplication, the operands can be divided into number of parts. The number of multiplications increase when we further decrease the chunks size of the operands. In this thesis we have optimize hardware resources and optimize simultaneously computation time. Utilizing the School-book multiplier and Karatsuba-Ofman algorithm (KA) to splitting the operands into two parts and then applied the same technique to divide the operands into four parts and eight parts. We have worked on three method i.e. two-part spiting, four-part and eight parts splitting. After the splitting of operands, these operands utilize the integer multiplier architecture and the Montgomery modular Multiplier algorithm. In the Montgomery modular Multiplication algorithm, the most important operation is integer multiplier architecture. Therefore, increase the speed of integer multiplier can help to enhance the overall efficiency of Montgomery modular Multiplier. The proposed design is simulated, synthesized and implemented using Xilinx ISE Design Suite by targeting different Xilinx FPGA devices for different bit sizes (64-1024). The proposed design of Montgomery modular multiplier is evaluated on the bases of computational time, area consumption, and throughput. It surpass the state of the art.

3.1. Integer Multipliers

In this algorithm multiplication complexity is overcome by dividing the operands into equivalent small chunks. A school-book multiplication complexity is Ō(n^2). The strategy found by Karatsuba-Ofman [29] dividing the operand into parts to decrease the complexity to Ō$(n^{1.58})$. In Two numbers multiplication A and B, the parts splitting algorithm suggested to divide these into the higher and lower parts as given below:

$$A_1 = \left(a_{n-1} \dots \dots \dots \dots a_{[n/2]}\right)$$

$$A_0 = \left(a_{[n/2]-1} \dots \dots \dots \dots a_0\right)$$

$$B_1 = \left(b_{n-1} \dots \dots \dots \dots b_{[n/2]}\right)$$

$$B_0 = \left(b_{[n/2]-1} \dots \dots \dots \dots b_0\right)$$

The operand A and B can be written as:

$$A = A_0 + 2^n A_1$$

$$B = B_0 + 2^n B_1$$

Now the multiplication result is given below:

$$Output = A\ x\ B = A_0 B_0 + 2^n (A_0 B_1 + A_1 B_0) + 2^{2n} A_1 B_1 \qquad (1)$$

Four multiplications are required if schoolbook method is adopted as show in above equation. They utilize the four DSP block for multiplication. The result of student book multiplier is fast but the use the resources.

However, using Karatsuba technique, the required number of multiplications are three as show in equation below:

$$Output = A\ x\ B = A_0 B_0 + 2^n \left(A_1 B_1 + A_0 B_0 - (A_1 - A_0)(B_1 - B_0)\right) + 2^{2n} A_1 B_1 \qquad (2)$$

The number of multiplications is required to multiply the two operands in the Karatsuba algorithm, which can save one multiplication operation and increase the adders and subtraction. They also utilized the sign bit operations. Which decrease the speed of the multiplication but they utilized the less resources of hardware. The repeated division of operands into the small parts until reaching the required size of the operand. Utilizing the school-book multiplier improves the speed of multiplication and Karatsuba algorithm use the less resources of hardware.

3.1.1. Operands Splitting

Operands may be divided into any number of parts and utilize the School-book (SB) or Karatsuba technique for the multiplication. The Table 2 shows the number of multipliers required according to operand parts with two different techniques of multiplication.

When we further divide, the operand size is decreasing, but the number of a multiplier is also increased with the addition of adder and subtraction to produce the last output. Further part of the operand may not be suitable because the area of the hardware is increased.

Table 2: Comparison of size and Multipliers for KA and SB

	School-Book (SB) Method		**Karatsuba Algorithm (KA)**	
Division	**Size**	**Multiplier**	**Size**	**Multiplier**
2-Part	N/2	4	N/2	3
3-Part	N/3	9	N/3	6
4-Part	N/4	16	N/4	10
5-Part	N/5	25	N/5	15

3.1.2. Two-parts splitting

Karatsuba algorithm says operands are split into two parts. Calculate the difference of divided operands $(A_1 - A_0)\ and (B_1 - B_0)$, it is the effective and vital part of the Karatsuba calculation. Then compute the product of the divided operand$(A_1 - A_0)\ and (B_1 - B_0)$. The result of these two operations is $\left(A_1B_1 + A_0B_0 - (A_1 - A_0)(B_1 - B_0)\right)$which is saved in one multiplier this is the beauty of Karatsuba algorithm (KA). Latest FPGA devices contain DSP blocks which consist of dedicated multipliers. In Xilinx ISE design suite Virtex-5 and Virtex-6 DSP blocks consist of 18x25 asymmetrical signed dedicated multiplier. The size of the dedicated multiplier is n-bit the output of the multiplication is 2n-bits. Which can use the three dedicated multipliers for the 2n-bits multiplication, the hardware architecture utilized the three dedicated multipliers. Operand size is less than 2n-bits it always utilizes the three dedicated multipliers in the hardware architecture. It is a generalized theory of multiplication. The old FPGA devices do not support a dedicated multiplier. The latest FPGA devices are fast for the multiplication due to the dedicated multiplier inside the DSP blocks which also perform the addition inside the dedicated multiplier.

3.1.3. Four-parts splitting

Repeatedly school-book multiplier and Karatsuba-Ofman calculation got applied on the operands. Four sections obtain after the applied splitting operation calculation on the two parts of the operands. The four sections of the operands are given below:

$$A = A_0 + 2^{n}A_1 + 2^{2n}A_2 + 2^{3n}A_3$$

$$B = B_0 + 2^{n}B_1 + 2^{2n}B_2 + 2^{3n}B_3$$

The general output of the multiplication is given below:

$$Output = A * B = A_0B_0 + 2^n(A_0B_1 + A_1B_0)$$

$$+2^{2n}(A_2B_0 + A_1B_1 + A_0B_2) + 2^{3n}(A_3B_0 + A_2B_1 + A_1B_2 + A_0B_3)$$

$$+2^{4n}(A_3B_1 + A_2B_2 + A_1B_3) + 2^{5n}(A_2B_3 + A_3B_2) + 2^{6n}A_3B_3 \quad (3)$$

The above condition demonstrates that 16 multiplications and 15 adders are needed for the output of the equation. The size of operand part is equal to DSP block size than we needed 16 dedicated multipliers to perform the multiplication of the equation. If restricted in the size of the multiplication, which uses only one DSP block than completion of task relies on performing it with 16 DSP blocks. After applying the Karatsuba-Ofman calculation on the same equation the output is:

$$Output = A * B = P_{00} + 2^n(P_{11} + P_{00} - D_{10})$$

$$+2^{2n}(P_{22} + P_{11} + P_{00} - D_{20}) + 2^{3n}(P_{33} + P_{22} + P_{11} + P_{00} - D_{30} - D_{21})$$

$$+2^{4n}(P_{33} + P_{22} + P_{11} - D_{31}) + 2^{5n}(P_{33} + P_{22} - D_{32}) + 2^{6n}(P_{33}) \quad (4)$$

After comparing both equations, it is observed that reduction occur in the number of multiplications from 16 to 10 and increase the 15 adder and 18 subtractions. The operands size remains the same one-fourth of the first operands. If the Single multiplication utilize the single DSP block than 10 numbers of DSP block are acquired to do complete multiplication. 6 numbers of the DSP block saved through Karatsuba-Ofman calculation. These numbers of operations utilize in the main equation:

$$P_{00} = A_0B_0$$

$$P_{11} = A_1B_1$$

$$P_{22} = A_2B_2$$

$$P_{33} = A_3B_3$$

$$D_{10} = (A_1 - A_0)(B_1 - B_0)$$

$$D_{20} = (A_2 - A_0)(B_2 - B_0)$$

$$D_{30} = (A_3 - A_0)(B_3 - B_0)$$

$$D_{21} = (A_2 - A_1)(B_2 - B_1)$$

$$D_{31} = (A_3 - A_1)(B_3 - B_1)$$

$$D_{32} = (A_3 - A_2)(B_3 - B_2)$$

3.1.4. Eight Part Splitting

Repeatedly school-book multiplier calculation got applied on the operands. Eight sections obtain after the applied splitting operation calculation on the two parts of the operands. The eight sections of the operands are given below:

$$\mathrm{A} = \mathrm{A}_0 + 2^{\mathrm{n}}\mathrm{A}_1 + 2^{2\mathrm{n}}\mathrm{A}_2 + 2^{3\mathrm{n}}\mathrm{A}_3 + 2^{4\mathrm{n}}\mathrm{A}_4 + 2^{5\mathrm{n}}\mathrm{A}_5 + 2^{6\mathrm{n}}\mathrm{A}_6 + 2^{7\mathrm{n}}\mathrm{A}_7$$

$$\mathrm{B} = \mathrm{B}_0 + 2^{\mathrm{n}}\mathrm{B}_1 + 2^{2\mathrm{n}}\mathrm{B}_2 + 2^{3\mathrm{n}}\mathrm{B}_3 + 2^{4\mathrm{n}}\mathrm{B}_4 + 2^{5\mathrm{n}}\mathrm{B}_5 + 2^{6\mathrm{n}}\mathrm{B}_6 + 2^{7\mathrm{n}}\mathrm{B}_7$$

The general output of the multiplication is given below:

$$A \; x \; B = A_0B_0 + 2^n(A_0B_1 + A_1B_0) + 2^{2n}(A_2B_0 + A_1B_1 + A_0B_2) + 2^{3n}(A_3B_0 + A_2B_1 + A_1B_2 + A_0B_3) + 2^{4n}(A_4B_0 + A_3B_1 + A_2B_2 + A_1B_3 + A_0B_4) + 2^{5n}(A_5B_0 + A_4B_1 + A_3B_2 + A_2B_3 + A_1B_4 + A_0B_5) + 2^{6n}(A_6B_0 + A_5B_1 + A_4B_2 + A_3B_3 + A_2B_4 + A_1B_5 + A_0B_6) + 2^{7n}(A_7B_0 + A_6B_1 + A_5B_2 + A_4B_3 + A_3B_4 + A_2B_5 + A_1B_6 + A_0B_7) + 2^{8n}(A_7B_1 + A_6B_2 + A_5B_3 + A_4B_4 + A_3B_5 + A_2B_6 + A_1B_7) + 2^{9n}(A_7B_2 + A_6B_3 + A_5B_4 + A_4B_5 + A_3B_6 + A_2B_7) + 2^{10n}(A_7B_3 + A_6B_4 + A_5B_5 + A_4B_6 + A_3B_7) + 2^{11n}(A_7B_4 + A_6B_5 + A_5B_6 + A_4B_7) + 2^{12n}(A_7B_5 + A_6B_6 + A_5B_7) + 2^{13n}(A_7B_6 + A_6B_7) + 2^{14n}A_7B_7 \quad (5)$$

The above condition demonstrates that 64 multiplications and 63 adders are needed for the output of the equation. The size of operand part is equal to DSP block size than we needed 64 dedicated multipliers to perform the multiplication of the equation. If restricted in the size of the multiplication, which uses only one DSP block than completion of task relies on performing it with 64 DSP blocks.

3.2. Montgomery Algorithm

In [28] a strategy for quickly measured multiplication has presented in 1985 by Peter L. Montgomery. Montgomery modular multiplication architecture measured A x B mod M, where A and B are certain whole numbers and M is a large prime number. Regular methodologies for processing the remainder of the division task is the cost. In Montgomery multiplication shift and adds operations replacing the costly of the division operation. Shift and adds operations strategy work only in the Montgomery domain. Before the task, the operand first transformed into the domain of Residue Number System. After completing the operation, the output is re-transformed. Word length must be selected in the power of two in the selection of radix R and modulus must be smaller than radix R. For the run of algorithm R and M must be a prime number. Whether for M (modulus) n bit is a positive number of A and B are two n bit operands. In modular multiplication, Output = A x B mod M where **0 < A; B < M.**

3.3. FPGA Implementation

Integrated circuits such as FPGA (Field Programmable Gate Arrays) could be programmed by user after fabrication. FPGA devices contain CLB (Configurable Logic Blocks) which are associated through programmable interconnects. This ability of the FPGA devices has made it for suitable hardware accelerators for different applications and they are largely deployed in cryptographic applications. The modern FPGA devices provide the dedicated portion for software and hardware cores and configurable blocks. In modern Xilinx FPGAs different memory, cores and dedicated blocks for the arithmetic operations are available which are already tailored for high speed and low power application. These cores are easily changeable according to requirement and utilize the multiple blocks at the same time. Configurable Logic Blocks provide the facility to the programmer to minimize the code to maximize the speed of the hardware.

CHAPTER 4.

Implementation and Results

This chapter deals with the brief description of implementation and the results of our proposed methodology.

4.1. Integer Multiplier

The Performance of the Montgomery modular multiplier totally depends on the integer multiplier efficiency. In this thesis, we have deployed school-book multiplier and Karatsuba-Ofman algorithm to increase the performance of the integer multiplier (IM). We adopt the three approaches to increase the efficiency of integer multiplier i.e. two-part splitting four-part splitting and eight part splitting. They are discussed with their results.

4.1.1. Karatsuba-Ofman Two-part Splitting Multiplier

Figure 4-1 show the architecture for two-part splitting multiplier and algorithm-1 describe the steps involved in synthesized of final result. In two-part splitting algorithm the operands can be split into two equal parts using Karatsuba-Ofman algorithm. In Figure 4-1 at the start of the multiplication, the input operands can store in the register A and B. In the next step, divide the operand into two part using Karatsuba-Ofman algorithm than generate the product using Integer multiplier. Two unsigned multipliers of N/2 bits and one signed multiplier of (N/2 + 1) bits to generate the third partial product. Three multiplication executed in parallel with the help of multiplier.

In algorithm-1 steps 1 to 12 explain the generation of partial product. In equation 2 the output utilizes the three multipliers. The reduction of multiplier is achieved through Karatsuba-Ofman algorithm. In the algorithm-1 step 13 to 15 show that the result of partial product will utilize as the inputs for the adders. The N/2 bits fast carry chain adders add the partial product shown in figure 4-1. Step 16 is the final addition to generate the product. The splitting depth of the operand is 1.

Table-2 shows the clock cycle for the two-part splitting multiplier. The complete product takes seven-clock cycle. In the figure 4-1 the first operation is to load the operand into the input register. In the Table-2, the LR show the load register which required one clock cycle.

The second operation in the Table-2 is computation of the Partial product, which is PPM Partial product multiplication, which utilized one clock cycle. The third operation in the figure is PPA partial product addition, which add the partial product in four-clock cycle. PPA is final stage. Final addition FA of the product, which add the PPA in one clock cycle. The whole multiplication utilized the seven-clock cycle for full product.

Table 2: Clock Cycle Two Parts Splitting Multiplication

Clock Cycle		**Task**
1^{st}	LR	Load Register
2^{nd}	PPM	Partial Product Multiplication
3^{rd} -6^{th}	PPA	Partial Product Addition
7^{th}	FA	Final Addition

Algorithm 1: Two Parts Splitting Multiplication Algorithm

Input A, B

$A = \sum_{k=0}^{1} 2^{ik} A_i$

$B = \sum_{k=0}^{1} 2^{ik} B_i$

Output Out=A x B

for i = 1; i ≥ 0; i = i-1 do

  j = i-1

  $P_{i,i} \leftarrow A_i \times B_i$

  while j ≥ 0 do

    $D_{i,i} \leftarrow (A_i - A_j) \times (B_i - B_j)$

    j ← j – 1

  end

End

$S_0 \leftarrow P_{00}$

$S_1 \leftarrow P_{11} + P_{00} - D_{10}$

$S_2 \leftarrow P_{11}$

$Out \leftarrow \sum_{k=0}^{2} 2^{ik} S_i$

return Out

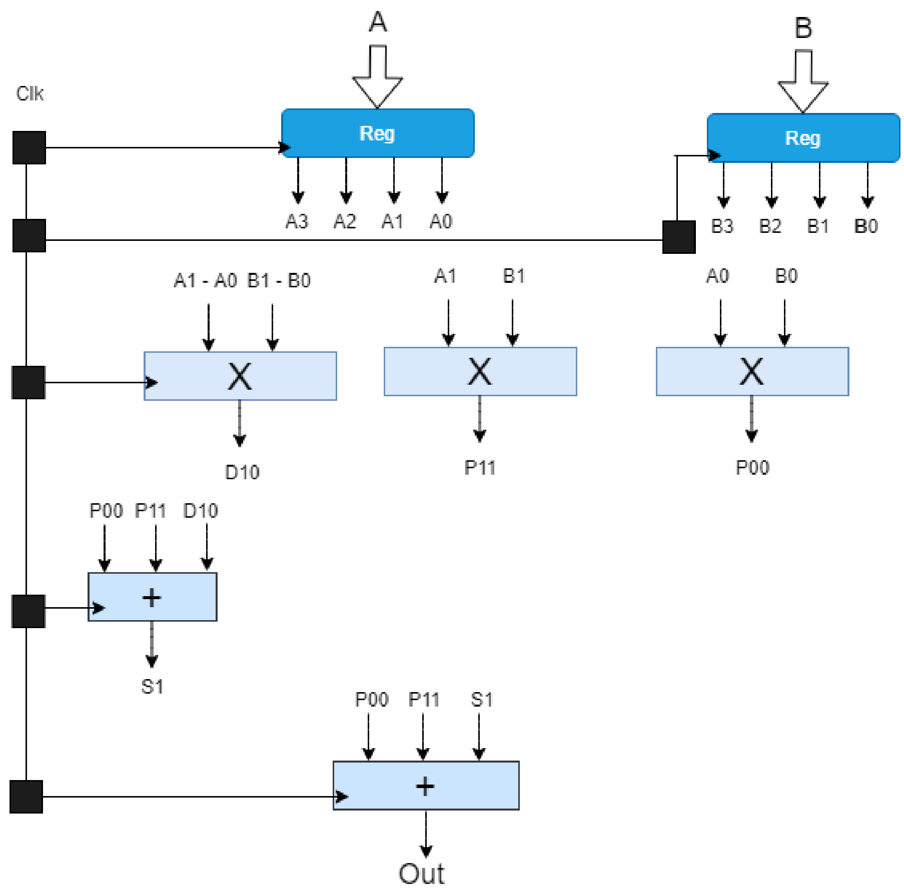

Figure 4-1: Two Parts Splitting Multiplication

4.1.2. Karatsuba-Ofman Four-part Splitting Multiplier

In four parts splitting method, the splitting depth is two. It means that each part of operand is further divided into two parts. The main advantage of four-part splitting multiplier is to optimize the hardware resources. Four-part splitting multiplier utilized the basic multiplier of DSP block in Virtex-6.

Figure 4-2 show the architecture for four-part splitting multiplier and algorithm-2 describe the steps involved in synthesized of final result. In four-part splitting algorithm, the operands can be split into four equal parts using to Karatsuba-Ofman algorithm. In Figure 4-

2 at the start of the multiplication, the input operands can store in the register A and B. In the next step, divide the operand into four part using Karatsuba-Ofman algorithm than generate the product using Integer multiplier. Four unsigned multipliers of N/4 bits and six signed multipliers of (N/4 + 1) bits to generate the ten partial products. Ten multiplication executed in parallel with the help of multiplier. In algorithm-2 steps 1 to 12 explain the generation of partial product. In equation-4 the output utilizes the ten multipliers. The reduction of multipliers is achieved through Karatsuba-Ofman algorithm. In the algorithm-2 step 13 to 19 show that the result of partial product utilizes as the inputs for the adders. The N/4 bits fast carry chain adders add the partial product shown in figure 4-2. Step 20 is the final addition to generate the product. The splitting depth of the operand is two.

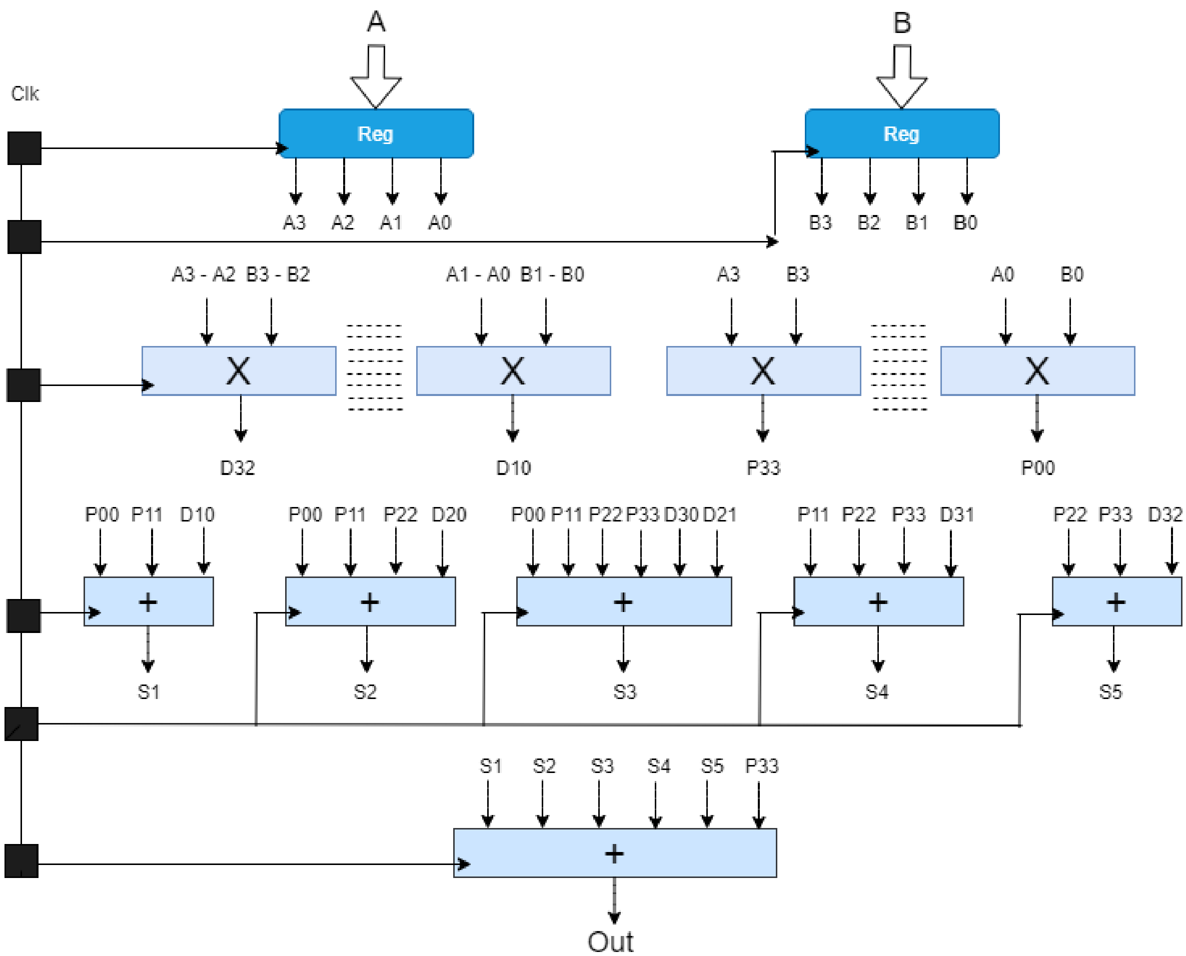

Figure 4-2: Four Parts Splitting Multiplication

Algorithm 2: Four Parts Splitting Multiplication Algorithm

Input A,B
$A = \sum_{k=0}^{3} 2^{ik} A_i$
$B = \sum_{k=0}^{3} 2^{ik} B_i$
Output Out=A x B
for i = 3 ; i ≥ 0 ; i = i-1 do
    j = i-1
    $P_{i,i} \leftarrow A_i \times B_i$
    while j ≥ 0 do
        $D_{i,i} \leftarrow (A_i - A_j) \times (B_i - B_j)$
        j ← j – 1
    end
end
$S_0 \leftarrow P_{00}$
$S_1 \leftarrow P_{11} + P_{00} - D_{10}$
$S_2 \leftarrow P_{22} + P_{11} + P_{00} - D_{10} - D_{20}$
$S_3 \leftarrow P_{33} + P_{22} + P_{11} + P_{00} - D_{30} - D_{21}$
$S_4 \leftarrow P_{22} + P_{11} + P_{33} - D_{31}$
$S_5 \leftarrow P_{22} + P_{33} - D_{32}$
$S_6 \leftarrow P_{11}$
$out \leftarrow \sum_{k=0}^{6} 2^{ik} S_i$
return Out

Table 3: Karatsuba-Ofman algorithm Two Part and Four-Part Splitting Multiplier

	Size Bits	LUT -	DSPs -	SR -	F MHz	CC -	Period Ns	Time ns	TP GOPS
Virtex 5	**KA Two Part Splitting Multiplier**								
	64	387	12	395	126.22	7	7.92	55.46	0.018
	128	1303	36	967	85.43	7	11.70	81.93	0.012
	256	5246	168	1927	65.66	7	15.23	106.62	0.009
	512	22003	693	3870	45.18	7	22.14	154.95	0.006
	KA Four Part Splitting Multiplier								
	64	744	11	317	246.71	7	4.05	28.37	0.035
	128	1941	40	1751	126.22	7	7.92	55.46	0.018
	256	5797	120	4079	85.43	7	11.70	81.93	0.012
	512	21719	560	8111	65.66	7	15.23	106.62	0.009
Virtex 6	**KA Two Part Splitting Multiplier**								
	64	387	12	395	133.37	7	7.50	52.49	0.019
	128	1303	36	967	95.87	7	10.43	73.02	0.014
	256	5246	168	1927	76.53	7	13.07	91.46	0.011
	512	21967	693	3870	54.73	7	18.27	127.90	0.008
	KA Four Part Splitting Multiplier								
	64	777	10	351	277.66	7	3.60	25.21	0.040
	128	1877	40	1686	133.37	7	7.50	52.49	0.019
	256	5669	120	3950	95.87	7	10.43	73.02	0.014
	512	21463	560	7854	76.53	7	13.07	91.46	0.011
Virtex 7	**KA Two Part Splitting Multiplier**								
	64	387	12	395	155.42	7	6.43	45.04	0.022
	128	1303	36	967	109.08	7	9.17	64.17	0.016
	256	5246	168	1927	88.01	7	11.36	79.54	0.013
	512	21967	693	3870	63.93	7	15.64	109.49	0.009
	1024	86748	2394	7731	42.41	7	23.58	165.07	0.006
	KA Four Part Splitting Multiplier								
	64	777	10	351	297.49	7	3.36	23.53	0.042
	128	1877	40	1686	155.42	7	6.43	45.04	0.022
	256	5669	120	3950	109.08	7	9.17	64.17	0.016
	512	21463	560	7854	88.01	7	11.36	79.54	0.013
	1024	93071	2310	15706	63.93	7	15.64	109.49	0.009

SR: Slice Registers **f:** Frequency **CC:** Clock Cycle **TP:** Throughput

GOPS: Giga Operation per Second **ns:** Nano Seconds

Table 4: Clock Cycle four Parts Splitting Multiplication

Clock Cycle		Task
1st	LR	Load Register
2nd	PPM	Partial Product Multiplication
3rd -6th	PPA	Partial Product Addition
7th	FA	Final Addition

Table-4 shows the clock cycle for the four-part splitting multipliers. The complete product takes seven-clock cycle. In the figure 4-2, the first operation is to load the operand into the input register. In the Table-4, the LR show the load register which required one clock cycle. The second operation in the Table-4 is computation of the Partial product, which is PPM partial product multiplication. Which utilized one clock cycle. The third operation in the figure is PPA partial product addition, which add the partial product in four-clock cycle. PPA is final stage. Final addition FA of the product, which add the PPA in one clock cycle. The whole multiplication utilized the seven-clock cycle for full product.

4.1.3. School-Book Two-part Splitting Multiplier

In School-book algorithm the operands can be split into two equal parts. The start of the multiplication, the input operands can store in the register A and B. In the next step, divide the operand into two part using spiting algorithm than generate the product using Integer multiplier. Two unsigned multipliers of N/2 bits. Four multiplication executed in parallel with the help of multiplier. In equation-1, the output utilizes the four multipliers.

Table-5 shows the clock cycle for the two-part splitting multiplier. The complete product takes two-clock cycle. The first operation is to load the operand into the input register. The LR show the load register which required one clock cycle. The second operation is Final addition FA of the product, which add the PPA in one clock cycle. The whole multiplication utilized the two-clock cycle for full product.

Table 5: Clock Cycle School-Book Multiplication

Clock Cycle		**Task**
1st	LR	Load Register
2nd	FA	Final Addition

4.1.4. School-Book four-part Splitting Multiplier

In four parts splitting method, the splitting depth is two. It means that each part of operand is further divided into two parts. The main advantage of four-part splitting multiplier is to optimize the hardware resources. Four-part splitting multiplier utilized the basic multiplier of DSP block in Virtex-6.

In four-part splitting algorithm, the operands can be split into four equal parts using to spiting algorithm. In the start of the multiplication, the input operands can store in the register A and B. In the next step, divide the operand into four part using splitting algorithm than generate the product using Integer multiplier. Sixteen unsigned multipliers of N/4 bits to generate the sixteen partial products. In equation-3, the output utilizes the sixteen multipliers. The splitting depth of the operand is two.

Table-5 shows the clock cycle for the four-part splitting multipliers. The complete product takes two-clock cycle. The first operation is to load the operand into the input register. In the Table-5, the LR show the load register which required one clock cycle. The second operation is Final addition FA of the product, which add the PPA in one clock cycle. The whole multiplication utilized the two-clock cycle for full product.

4.1.5. School-Book Eight-part Splitting Multiplier

In Eight parts splitting method, the splitting depth is three. It means that each part of operand is further divided into two parts. The main advantage of Eight-part splitting multiplier is to optimize the hardware resources. Four-part splitting multiplier utilized the basic multiplier of DSP block in Virtex-6 and virtex7.

In Eight-part splitting algorithm, the operands can be split into eight equal parts using to spiting algorithm. In the start of the multiplication, the input operands can store in the register A and B. In the next step, divide the operand into eight part using splitting algorithm than generate the product using Integer multiplier. Sixty-four unsigned multipliers of N/8 bits to generate the sixty-four partial products. In equation-5, the output utilizes the sixty-four multipliers. The splitting depth of the operand is three.

Table 6: School-Book algorithm Two, Four and Eight-Part Splitting Multiplier

	Size bits	LUT -	DSPs -	SR -	F MHz	CC -	Period ns	Time ns	TP GOPS
Virtex 5	**SB Two Part Splitting Multiplier**								
	64	255	16	330	126.215	2	7.92	15.85	0.063
	128	1139	48	899	85.434	2	11.70	23.41	0.043
	256	5135	224	1794	65.655	2	15.23	30.46	0.033
	512	19691	924	3608	45.177	2	22.14	44.27	0.023
	SB Four Part Splitting Multiplier								
	64	560	16	585	427.332	2	2.34	4.68	0.214
	128	1151	64	1518	126.215	2	7.92	15.85	0.063
	256	4815	192	3983	85.434	2	11.70	23.41	0.043
	512	21055	896	7950	65.655	2	15.23	30.46	0.033
	SB Eight Part Splitting Multiplier								
	64	1168	64	1169	506.047	2	1.98	3.95	0.253
	128	2336	64	2289	427.332	2	2.34	4.68	0.214
	256	4703	256	6270	126.215	2	7.92	15.85	0.063
	512	19455	768	16319	85.434	2	11.70	23.41	0.043

Virtex 6	**SB Two Part Splitting Multiplier**								
	64	255	16	330	133.369	2	7.50	15.00	0.067
	128	1139	48	912	95.865	2	10.43	20.86	0.048
	256	5135	224	1794	76.533	2	13.07	26.13	0.038
	512	19619	924	3608	54.73	2	18.27	36.54	0.027
	SB Four Part Splitting Multiplier								
	64	560	16	585	533.86	2	1.87	3.75	0.267
	128	1151	64	1518	133.369	2	7.50	15.00	0.067
	256	4815	192	3996	95.865	2	10.43	20.86	0.048
	512	21055	896	7950	76.533	2	13.07	26.13	0.038
	SB Eight Part Splitting Multiplier								
	64	1143	65	1052	436.462	2	2.29	4.58	0.218
	128	2336	64	2289	533.86	2	1.87	3.75	0.267
	256	4703	256	6270	133.369	2	7.50	15.00	0.067
	512	19455	768	16332	95.865	2	10.43	20.86	0.048
Virtex 7	**SB Two Part Splitting Multiplier**								
	64	255	16	330	155.424	2	6.43	12.87	0.078
	128	1139	48	898	109.077	2	9.17	18.34	0.055
	256	5135	224	1794	88.009	2	11.36	22.72	0.044
	512	19619	924	3608	63.93	2	15.64	31.28	0.032
	1024	67403	3192	7214	42.405	2	23.58	47.16	0.021
	SB Four Part Splitting Multiplier								
	64	560	16	585	568.134	2	1.76	3.52	0.284
	128	1151	64	1518	155.424	2	6.43	12.87	0.078
	256	4815	192	3982	109.077	2	9.17	18.34	0.055
	512	21055	896	7950	88.009	2	11.36	22.72	0.044
	1024	79503	3696	15930	63.93	2	15.64	31.28	0.032
	SB Eight Part Splitting Multiplier								
	64	1052	65	1143	480.273	2	2.08	4.16	0.240
	128	2336	64	2289	568.134	2	1.76	3.52	0.284
	256	4703	256	6270	155.424	2	6.43	12.87	0.078
	512	19455	768	16318	109.077	2	9.17	18.34	0.055
	1024	84607	3584	32574	88.009	2	11.36	22.72	0.044

SR: Slice Registers **f:** Frequency **CC:** Clock Cycle **TP:** Throughput

GOPS: Giga Operation per Second **ns:** Nano Seconds

4.2. Montgomery Modular Multiplier Architecture

The architecture of Montgomery modular multiplier (MMM) is shown in algorithm-3. In this algorithm, there are three n-bit Integer multiplier. The overall efficiency of the Montgomery multiplier algorithm is dependent on the Integer multiplier. In this thesis, we present an efficient Montgomery modular multiplier (MMM) implemented on modern FPGA devices.

The proposed architecture of Montgomery multiplier is shown in figure 4-3. This architecture consists of n-bit integer multiplier. The intermediate multiplication results holding in 2n-bit register. The holding result in register is utilized in next steps. All the three multiplication are executed in series. In this architecture the result of the first integer multiplication is stored in the register. The stored result than got added in the third multiplication result. The final step is reduction that utilize to compute the Montgomery multiplication.

Algorithm 3: Montgomery Multiplier

$Input: A, B, M, n = \log_2 M, R = 2^n$

$M_1 = -M^{-1} \, mod \, R$

$Result: Output = A \times B \times R^{-1} \, mod \, M$

$D \leftarrow A \times B$

$E \leftarrow D \times M_1 \, mod \, R$

$Output \leftarrow (D + E \times M)/R$

$If \; Output > M \; then \; return \; Output - M$

$Else \; return \; Output$

In the Table-7 the proposed architecture, performs the series of operation for executing the Montgomery modular multiplication. In the first clock cycle the operands is loaded in Register A and B which is represented by Load Register (L). The first integer multiplication operation utilizes the input operands A and B. When the multiplier gets the operands A and B, the multiplication operation started. It consumes the seven-clock cycle in Karatsuba Algorithm and two-clock cycle in School-book Algorithm to compute the multiplication of the operands and 2n-bit product result stored in the register, which is represented by W in Table-7. The result of the product written in the register file utilize the one clock cycle. Table-7 show that three multiplication were executed in series.

During the operation Load Register new operands got loaded in the input register A and B. The second multiplication utilize modulus of first multiplication output and M1 as operand. The second multiplication also utilize the seven-clock cycle in Karatsuba Algorithm and two-clock cycle in School-book Algorithm to compute the product and one-clock cycle required to store the result in register file. For the three-multiplication required twenty-six clock cycle in series in Karatsuba Algorithm and eleven-clock cycle in School-book Algorithm.

In the last step, three more clock cycle required for the addition of the product of the three multiplications according to the algorithm-3. Comparison and subtraction operation compute if required. In this way, Montgomery modular multiplication architecture required twenty-nine clock cycle in Karatsuba Algorithm and fourteen-clock cycle in School-book Algorithm to compute the complete result

Table 7: Karatsuba and School-Book Algorithm Clock Cycle for Montgomery Multiplier

Karatsuba Algorithm Clock Cycle	**Operations**		**School-Book Algorithm Clock Cycle**
1st	L	Load Register	1st
2nd-8th	IR	Integer Multiplication	2nd-3rd
9th	WR	Write to Register File	4th
10th	L	Load Register	5th
11th -17th	IR	Integer Multiplication	6th -7th
18th	WR	Write to Register File	8th
19th	L	Load Register	9th
20th-26th	IR	Integer Multiplication	10th-11th
27th	A	Addition	12th
28th	C	Comparison	13th
29th	S	Subtraction	14th

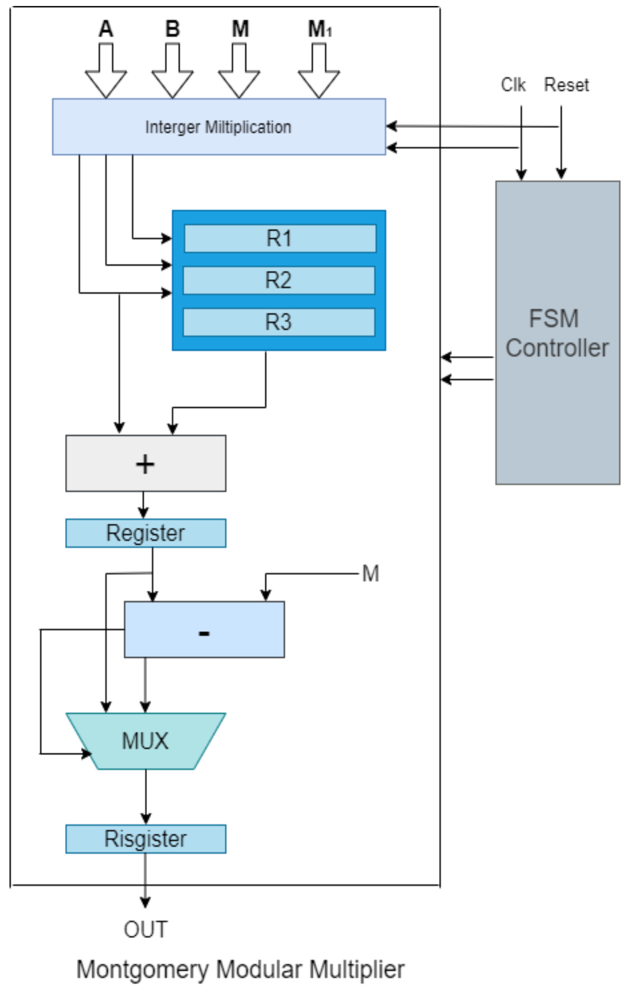

Figure 4-3: Montgomery Modular Multiplier for Integer multiplier

4.3. Implementation Results

In the previous sections, the proposed architectures are not specific design to FPGA family. The length of the operand at the root level of four-part splitting is selected for the implementation in serval FPGA devices.

In the proposed architectures of two-part splitting, four-part splitting and eight-part splitting are calculated for five common operands i.e. bit sizes 64,128,256,512 and 1024 . The proposed architectures of Montgomery modular multiplier have been designed in

Table 8: Montgomery Multiplier for Karatsuba Algorithm

	Size bits	LUT -	DSPs -	SR -	F MHz	CC -	Period ns	Time ns	TP GOPS
Virtex 5	**Montgomery Multiplier with KA Two Part Splitting Technique**								
	64	875	12	912	102.15	29	9.79	283.91	0.004
	128	2265	36	1996	68.96	29	14.50	420.52	0.002
	256	7155	168	3980	50.79	29	19.69	571.01	0.002
	Montgomery Multiplier with KA Four Part Splitting Technique								
	64	1232	11	834	225.01	29	4.44	128.89	0.008
	128	2903	40	2780	102.05	29	9.80	284.18	0.004
	256	7706	120	6132	68.94	29	14.50	420.64	0.002
	512	25523	560	12212	40.28	29	24.83	720.03	0.001
Virtex 6	**Montgomery Multiplier with KA Two Part Splitting Technique**								
	64	810	12	912	109.31	29	9.15	265.30	0.004
	128	2135	36	1996	79.05	29	12.65	366.84	0.003
	256	6680	168	3980	59.87	29	16.70	484.36	0.002
	512	24782	693	7971	43.70	29	22.89	663.69	0.002
	Montgomery Multiplier with KA Four Part Splitting Technique								
	64	1200	10	868	277.66	29	3.60	104.44	0.010
	128	2709	40	2715	109.17	29	9.16	265.65	0.004
	256	7103	120	6003	78.99	29	12.66	367.16	0.003
	512	24278	560	11955	53.66	29	18.64	540.43	0.002
Virtex 7	**Montgomery Multiplier with KA Two Part Splitting Technique**								
	64	810	12	912	123.22	29	8.12	235.36	0.004
	128	2135	36	1996	88.12	29	11.35	329.11	0.003
	256	6680	168	3980	66.23	29	15.10	437.85	0.002
	512	24782	693	7971	48.64	29	20.56	596.20	0.002
	1024	92328	2394	16162	33.84	29	29.55	856.87	0.001
	Montgomery Multiplier with KA Four Part Splitting Technique								
	64	1200	10	868	297.49	29	3.362	97.48	0.010
	128	2709	40	2715	123.04	29	8.128	235.71	0.004
	256	7103	120	6003	88.03	29	11.360	329.43	0.003
	512	24278	560	11955	66.19	29	15.109	438.17	0.002
	1024	98651	2310	23903	34.62	29	28.888	837.74	0.001

SR: Slice Registers **f:** Frequency **CC:** Clock Cycle **TP:** Throughput

GOPS: Giga Operation per Second **ns:** Nano Seconds

Verilog hardware description language and synthesis has been done in Xilinx ISE Design Suite 14.1 on different devices (Virtex-5, Virtex-6, Virtex-7). Table-8 shows the implemented results for two part and four-part splitting Montgomery modular multiplier in FPGA device. Table 9 shows the implemented results for two part, four and eight-Part splitting Montgomery modular multiplier for school-book. The table shows that the eight-part splitting architectures utilize the less DSP blocks for same operand length comparatively to two-part splitting and four-part splitting. The advantage of saving DSP blocks is only for higher operand length. The reason behind the saving DSP block is chunks the length of the operands have. When the operands splitting depth is two and after splitting the length of the operands is less than the length of the multiplier provided by the DSP blocks then Montgomery modular multiplier implementation results show that the time increased in Eight-part splitting method.

Table 9: Montgomery Multiplier for School-Book Algorithm

	Size	**LUT**	**DSPs**	**SR**	**F**	**CC**	**Period**	**Time**	**TP**
	bits	**-**	**-**	**-**	**MHz**	**-**	**ns**	**ns**	**GOPS**
Virtex 5	**Montgomery Multiplier with SB Two Part Splitting Technique**								
	64	742	16	846	126.22	14	7.92	110.92	0.009
	128	2100	48	1927	85.43	14	11.70	163.87	0.006
	256	7043	224	3846	65.66	14	15.23	213.24	0.005
	512	118225	693	8209	40.27	14	24.83	347.67	0.003
	Montgomery Multiplier with SB Four Part Splitting Technique								
	64	1062	16	1026	255.01	14	3.92	54.90	0.018
	128	2112	64	2546	126.22	14	7.92	110.92	0.009
	256	6723	192	6034	75.87	14	13.18	184.52	0.005
	512	96512	728	12690	40.27	14	24.83	347.67	0.003
	Montgomery Multiplier with SB Eight Part Splitting Technique								
	64	1662	64	1594	255.01	14	3.92	54.90	0.018
	128	3312	64	3122	135.94	14	7.36	102.99	0.010
	256	6611	256	8322	75.87	14	13.18	184.52	0.005
	512	90456	636	21122	40.27	14	24.83	347.67	0.003
Virtex 6	**Montgomery Multiplier with SB Two Part Splitting Technique**								
	64	677	16	846	133.37	14	7.50	104.97	0.010
	128	1970	48	1940	95.87	14	10.43	146.04	0.007
	256	6566	224	3846	76.53	14	13.07	182.93	0.005
	512	22431	924	7708	53.66	14	18.64	260.90	0.004

	Montgomery Multiplier with SB Four Part Splitting Technique								
	64	977	16	1026	293.84	14	3.40	47.65	0.021
	128	1982	64	2546	133.37	14	7.50	104.97	0.010
	256	6246	192	6048	95.87	14	10.43	146.04	0.007
	512	23867	896	12050	53.66	14	18.64	260.90	0.004
	Montgomery Multiplier with SB Eight Part Splitting Technique								
	64	1565	65	1568	293.84	14	3.40	47.65	0.021
	128	3182	64	3122	179.24	14	5.58	78.11	0.013
	256	6134	256	8322	100.69	14	9.93	139.04	0.007
	512	22267	768	20418	53.66	14	18.64	260.90	0.004
Virtex 7	**Montgomery Multiplier with SB Two Part Splitting Technique**								
	64	677	16	846	155.42	14	6.43	90.08	0.011
	128	1970	48	1926	109.08	14	9.17	128.35	0.008
	256	6566	224	3846	88.01	14	11.36	159.07	0.006
	512	22431	924	7708	63.93	14	15.64	218.99	0.005
	1024	72980	3192	15410	34.62	14	28.89	404.43	0.002
	Montgomery Multiplier with SB Four Part Splitting Technique								
	64	997	16	1026	337.01	14	2.97	41.54	0.024
	128	1982	64	2546	155.42	14	6.43	90.08	0.011
	256	6246	192	6034	109.08	14	9.17	128.35	0.008
	512	23867	896	12050	66.39	14	15.06	210.89	0.005
	1024	85080	3696	24126	34.62	14	28.89	404.43	0.002
	Montgomery Multiplier with SB Eight Part Splitting Technique								
	64	1568	65	1568	337.01	14	2.97	41.54	0.024
	128	3182	64	3122	212.98	14	4.70	65.73	0.015
	256	6134	256	8322	122.68	14	8.15	114.12	0.009
	512	22267	768	20418	66.39	14	15.06	210.89	0.005
	1024	90184	3584	40770	34.62	14	28.89	404.43	0.002

SR: Slice Registers **f:** Frequency **CC:** Clock Cycle **TP:** Throughput

GOPS: Giga Operation per Second **ns:** Nano Seconds

Table 10 show that the performance differences with other design architectures on same platform. The proposed design of 256bit Montgomery modular multiplier architectures run at frequency 95.87 MHz and utilized fourteen clock cycles to compute result. The proposed design consumes 192 DSP blocks. The other proposed design of 256bit Montgomery modular multiplier architectures run at frequency 78.985 MHz and utilized twenty-nine clock cycles to compute result. The proposed design consumes 120 DSP blocks. Mondal et al. In [1] utilized the school-book architectures to compute the Montgomery modular multiplier. It put up the concept to consume 16 64x64 bit soft cores architecture operated at 102MHz frequency. Their resources regarding hardware architecture consumed double as compare to purposed design. 29% less frequency is achieved here.

Brinci et al. In [2] presented a concept for the multiplier of Barreto-Naehrig curves. They provided the special prime number for implementations of B-N curves and utilized the non-standard division for adjustment of the FPGA Digital Signal Processor (DSP) block. This architecture achieves 208 MHz frequency. However, they are only flexible with Barreto-Naehrig curves. This is the main drawback of this architecture. They utilized 50% more DSP block as compare to proposed architecture. K. Javeed [5] elaborated the concept with the addition of 512-bit and multiplication of 256-bit utilization of the carry chain of 64-bit with soft-core multiplier. They utilized the school multiplier in Montgomery modular multiplier and succeed to 188 MHz frequency. They utilized 200% more DSP block comparatively to proposed architecture. Yang et al. In [6] advanced concept of implementation scheme of using IP cores of the FPGA with the addition of 512-bit and 256-bit multiplication to achieve50% better results as compared to standard implementation. They improved the low frequency and high latency in this architecture.

In this architecture, they achieve 40MHz frequency, which is too low for high-speed applications. The drawback of this architecture is observed in schoolbook multiplier in Montgomery modular multiplier, which utilized the too much area. C. J. McIvor [7] presented concept to implement the hardware processor for ECC. Full block Montgomery modular multiplication with the 256-bit integer multiplier (IM) is intended to 16-bit cascading unsigned multipliers and this method is sustained until the specified size of the multiplier is achieved. Fast carry look-ahead adders are required for the addition of the modular multiplication. The drawback of this architecture in term of time is long duration for synthesis comparatively to our purposed architecture.

G. C. Chow [11] put up the design concept when FPGA working of high frequency and routing delay is increased in large multipliers. For decreasing the routing delay dividing the operands into small parts by using the Karatsuba algorithm and these parts are using the dedicated multipliers to increase the efficiency of the hardware with deep pipeline stages.

The number of pipeline stage and time for Montgomery modular multiplier is not mentioned in this paper. The drawback of this architecture is utilized more than 50 clock cycle. Our purposed architecture utilized only 29-clock cycle for one Montgomery modular multiplier.

Table 10: Performance Comparison

Design	**Device**	**Size Bits**	**Area LUT**	**DSPs**	**F MHz**	**CC**	**Period Ns**	**Time ns**	**TP GOPS**
Proposed SB Four Part Splitting Montomery Multiplier									
[Proposed]	Virtex-6	64	977	16	293.84	14	3.40	47.65	0.021
		128	1982	64	133.37	14	7.50	104.97	0.010
		256	6246	192	95.87	14	10.43	146.04	0.007
		512	23867	896	53.66	14	18.64	260.90	0.004
Proposed KA Four Part Splitting Montomery Multiplier									
[Proposed]	Virtex-6	64	1200	10	277.66	29	3.60	104.44	0.010
		128	2709	40	109.17	29	9.16	265.65	0.004
		256	7103	120	78.99	29	12.66	367.16	0.003
		512	24278	560	53.66	29	18.64	540.43	0.002
[1]	Virtex-6	256	-	256	102.67	-	9.74	-	-
[2]	Virtex-6	258	-	176	208.00	-	4.81	-	-
[5]	Virtex-6	256	-	256	188.00	42	5.32	223.40	0.004
[6]	Virtex-6	256	24000	256	40.06	50	24.96	1248.13	0.001
[7]	Virtex-6	256	1420	256	45.68	32	21.89	700.53	0.001
[10]	Virtex-6	256	3900	144	96.00	42	10.42	437.50	0.002
[11]	Virtex-6	256	17000	108	336.00	50	2.98	148.81	0.007
[13]	Virtex-6	256	22500	108	205.76	29	4.86	140.94	0.007
[14]	Virtex-6	256	3900	-	95.20	130	10.50	1365.55	0.001
[15]	Virtex-6	256	6300	-	166.00	132	6.02	795.18	0.001

SR: Slice Registers **f:** Frequency **CC:** Clock Cycle **TP:** Throughput

GOPS: Giga Operation per Second **ns:** Nano Seconds

I. San, N. At [12] presented the design concept of the Karatsuba algorithm to dividing the operand into two chunks and use in the Montgomery modular multiplication algorithm with higher radix. Architecture uses the dedicated blocks of the multiplier. The drawback of this architecture is increasing the space of the hardware. X. Yan [13] put up the design concept of the Karatsuba algorithm divided into 4 levels use in Montgomery modular multiplier. Using the splitting method operand is divided into two parts. The divided part again divided into two other parts in the reappearance style until the divided parts are length matches with the DSP blocks of an FPGA. Utilized the LUTs on FPGA rather than the dedicated multipliers. They utilized same number of clock cycle as in our purposed architecture. The results is better in terms of time. Our purposed architecture is better in term of utilization of hardware resources and time delay.

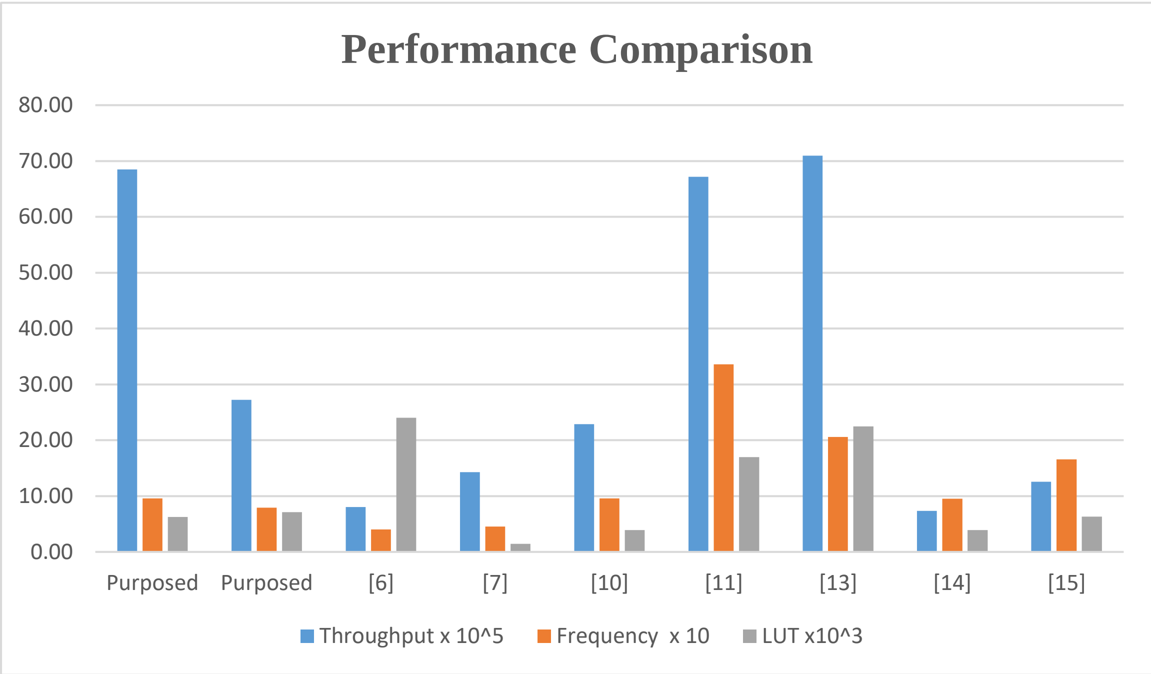

Figure 4-4 Performance Comparison

We also discussed the result of Montgomery modular multiplier with bit-wise implementation. In modern FPGA dedicated multipliers are not used in bitwise implementations, it uses only standard FPGA. The utilization of dedicated multipliers on FPGA is faster than the standard design-based FPGA. The purposed architecture in [17] consume the 256bit interleaved modular multiplier. In this architecture, they achieve 96MHz frequency.

The results of our purposed architecture are 4 time better in terms of time for Montgomery modular multiplier and throughput. K. Javeed [14] presented the design concept of LUT used on the hardware implementation rather than the FPGA dedicated multipliers. Radix-4 based on modular multiplier with the serial interleaved. In table 10, shown that our purposed architecture is 4 time better comparatively to design of K. Javeed [15] who presented the similar design concept of parallel interleaved modular multiplier implementation of hardware architecture. According to the architecture, an operand is working on four parallel processing elements to complete the dedicated task according to the algorithm. The results of our purposed architecture are 2.5 time better in terms of time period and throughput.

CHAPTER 5.

Conclusions

In the hardware implementation of public key cryptographic algorithms, Montgomery Modular Multiplier pay a vital role. This thesis provides a full-word implementation of Montgomery Modular multiplication which enhance the execution speed of Elliptic-curve cryptography (ECC) and RSA cryptographic algorithms on hardware. We have utilized the Karatsuba–Ofman and school-book algorithm to calculate the 64-1024 Bits.

Multiplications are done by using Xilinx FPGA devices. In this work we exploit the efficiency of Karatsuba-ofman (KA) and School-book (SB) algorithm to deploy a 1024-bit Montgomery modular multiplier architecture. We implement the Karatsuba–Ofman (KA) and Schook-book (SB) techniques to divide the operands on different size according to Xilinx FPGA devices. The proposed design is evaluated on computational time, area consumption, throughput and it will significantly surpass the state of the art.

REFERENCES

[1] A. Mondal, S. Ghosh, A. Das, D. R. Chowdhury, Efficient FPGA implementation of Montgomery multiplier using dsp blocks, in: Progress in VLSI Design and Test, Springer, 2012, pp. 370-372.

[2] R. Brinci, W. Khmiri, M. Mbarek, A. B. Raba^a, A. Bouallegue, F. Chekir, Efficient multiplier for pairings over barreto-naehrig curves on virtex-6 FPGA, IACR Cryptology EPrint Archive 2013 (2013) 5.

[3] S.-R. Kuang, K.-Y. Wu, R.-Y. Lu, Low-cost high-performance vlsi architecture for montgomery modular multiplication, IEEE Transactions on Very Large-Scale Integration (VLSI) Systems 24 (2) (2016) 434-443.

[4] A. Rezai, P. Keshavarzi, High-throughput modular multiplication and exponentiation algorithms using multibit-scan-multibit-shift technique, IEEE Transactions on Very Large

Scale Integration (VLSI) Systems 23 (9) (2015) 1710-1719.

[5] K. Javeed, X. Wang, Efficient Montgomery multiplier for pairing and elliptic curve-based cryptography, in: Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2014 9th International Symposium on, IEEE, 2014, pp. 255-260

[6] Y. Yang, C. Wu, Z. Li, J. Yang, Efficient FPGA implementation of modular multiplication based on Montgomery algorithm, Microprocessors and Microsystems 47 (2016) 209-215.

[7] C. J. McIvor, M. McLoone, J. V. McCanny, Hardware elliptic curve cryptographic processor over rmgf (p), IEEE Transactions on Circuits and Systems I: Regular Papers 53 (9) (2006) 1946-1957.

[8] M. Morales-Sandoval, A. Diaz-Perez, Scalable gf (p) Montgomery multiplier based on a digit-digit computation approach, IET Computers & Digital Techniques 10 (3) (2016) 102-109

[9] Y. Gong, S. Li, High-throughput FPGA implementation of 256-bit Montgomery modular multiplier, in: Education Technology and Computer Science (ETCS), 2010 Second International Workshop on, Vol. 3, IEEE, 2010, pp. 173-176.

[10] S. Ghosh, I. Verbauwhede, D. Roychowdhury, and Core based architecture to speed up optimal ate pairing on FPGA platform, in: International Conference on Pairing-Based Cryptography,

Springer, 2012, pp. 141-159.

[11] G. C. Chow, K. Eguro, W. Luk, P. Leong, A karatsuba-based Montgomery multiplier, in: Field Programmable Logic and Applications (FPL), 2010 International Conference on, IEEE, 2010, pp. 434-437.

[12] I. San, N. At, Improving the computational efficiency of modular operations for embedded systems, Journal of Systems Architecture 60 (5) (2014) 440-451.

[13] X. Yan, G. Wu, D. Wu, F. Zheng, X. Xie, An implementation of Montgomery modular multiplication on FPGA in: Information Science and Cloud Computing (ISCC), 2013 International Conference on, IEEE, 2013, pp. 32-38.

[14] K. Javeed, X. Wang, M. Scott, High performance hardware support for elliptic curve cryptography over general prime field, Microprocessors and Microsystems 51 (2017) 331-342.

[15] K. Javeed, X. Wang, Low latency exible FPGA implementation of point multiplication on elliptic curves over gf (p), International Journal of Circuit Theory and Applications 45 (2) (2017) 214-228.

[16] S. B. Ors, L. Batina, B. Preneel, J. Vandewalle, Hardware implementation of an elliptic curve processor over gf (p), in: Application-Speci c Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on, IEEE, 2003, pp. 433-443.

[17] K. Javeed, X. Wang, M. Scott, Serial and parallel interleaved modular multipliers on FPGA platform, in: Field Programmable Logic and Applications (FPL), 2015 25th International Conference on, IEEE, 2015, pp. 1-4.

[18] Francisco Rodriguez, N.A. Saquib et al, "Cryptographic Algorithms on Reconfigurable Hardware", Springer, 2006. pp 105-108

[19] E.F. Brickell, "A Fast-Modular Multiplication Algorithm with Application to Two key Cryptography," in Advances in Cryptology, Proceedings of Crypto 86, pages 51-60

[20] P.L. Montgomery, "Modular Multiplication without Trial Division," Mathematics of Computation, Vol. 44, 1985, pp 519-521.

[21] K. Posch and R. Posch, “Modulo Reduction in Residue Number Systems,” IEEE Trans. Parallel and Distributed Systems, Vol. 6, No. 5, pp. 449-454, 1995.

[22] C. McIvor, M. McLoone, and J. V. McCanny, “High-radix systolic modular multiplication on reconfigurable hardware,” in Proc. IEEE International Conference on Field-Programmable Technology 2005 (ICFPT’05), Dec. 2005, pp. 13–18.

[23] Elbert, A.J., and Paar, C. "Towards an FPGA Architecture Optimized for Public-Key Algorithms’. Presented at the SPIE Symposium on Voice, Video and Communications, Sept. 1999

[24] Miaoqing Huang, Kris Gaj, Tarek El-Ghazawi, "New Hardware Architectures for Montgomery Modular Multiplication Algorithm," in IEEE Transactions on Computers, Vol.60, Issue 7, July 2011.

[25] Miroslav Knezevic, Frederik Vercauteren, Ingrid Verbauwhede, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods "IEEE Transactions On Computers, Vol. 59, No. 12, December 2010.

[26] Buminov V. Schimmler, Manfred "Area and time efficient modular multiplication of large integers," Proceedings of IEEE International Conference on Application-Specific Systems, Architectures, and Processors, 24-26 June 2003.

[27] V. Buminov M. Schmimmer, “Area-Time Optimal Modular Multiplication,” Embedded Cryptographic Hardware: Methodologies and Architectures, 2004, ISBN 1594540128.

[28] P. L. Montgomery, Modular multiplication without trial division, Mathematics of computation 44 (170) (1985) 519-521.

[29] A. Karatsuba, Y. Ofman, Multiplication of many-digital numbers by automatic computers, in: Doklady Akad. Nauk SSSR,Vol. 145, 1962, p. 85

[30] P.M. Kogge and H.S. Stone, "A Parallel Algorithm for the Efficient Solution of a General Class of Recurrence Equations," IEEE Transactions on Computers, Vol. 22, No. 8, pp. 786-792, August 1973.

[31] Neil H.E. Weste, David Money Harris," CMOS VLSI Design: A Circuits and Systems Perspective, “Fourth Edition, Addison Wesley

[32]. K V Gowreesrinivas, P Samundiswary, “Comparative Study of Performance of Single Precision Floating Point Multiplier Using Vedic Multiplier and different types of Adders” Proceedings of International Conference on Control, Instrumentation, Communication and

Computational Technologies, Kanyakumari, Tamilnadu, pp.558-563, Dec 2016.

[33]. N Jitendra Babu, Rajkumar Sarma, “A Novel Low Power and High Speed Multiply Accumulate (MAC) unit Design for Floating-Point Numbers,” Proceedings of International Conference on Smart Technologies and Management for Computing Communication, Controls, Energy and Materials, Chennai, pp.411-411, May 2015.

[34]. Darjn Esposito, Davide De Caro,”Variable Latency Speculative Parallel Prefix Adders for Unsigned and Signed Operands”, IEEE Transactions On Circuits And Systems-I,Vol. 63, NO. 8, Aug 2016

[35]. Giorgos Dimitrakopoulos, Dimitris Nikolos, “High-Speed Parallel Prefix VLSI Ling Adders”, IEEE TRANSACTIONS ON COMPUTERS, VOL.54, NO.2, FEB 2005

[36] W. Diffie, M.E. Hellman, New directions in cryptography, IEEE Trans. Inf. Theory 22 (6) (2006) 644–654.

[37] R.L. Rivest, A. Shamir, L.M. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Commun. ACM 21 (2) (1978) 120–126.

[38] N. Koblitz, Elliptic curve cryptosystems, Math. Comput. 48 (177) (1987) 203–209.

[39] V.S. Miller, Use of elliptic curves in cryptography, in: H.C. Williams (Ed.), Advances in Cryptology - CRYPTO ’85 proceedings, 218 Springer-Verlag, 1986, pp. 417–426.

[40] P.L. Montgomery, Modular multiplication without trial division, Math. Comput. 44 (170) (1985) 519–521.

[41] A. Mondal, S. Ghosh, A. Das, D. R. Chowdhury, Efficient fpga implementation of montgomery multiplier using dsp blocks, in: Progress in VLSI Design and Test, Springer, 2012, pp. 370-372.