2012

Fast Modular Reduction for Large-Integer Multiplication

Suhas Sreehari
University of Windsor

Follow this and additional works at: https://scholar.uwindsor.ca/etd

Recommended Citation
Sreehari, Suhas, "Fast Modular Reduction for Large-Integer Multiplication" (2012). Electronic Theses and Dissertations. 5360.
https://scholar.uwindsor.ca/etd/5360

This online database contains the full-text of PhD dissertations and Masters’ theses of University of Windsor students from 1954 forward. These documents are made available for personal study and research purposes only, in accordance with the Canadian Copyright Act and the Creative Commons license—CC BY-NC-ND (Attribution, Non-Commercial, No Derivative Works). Under this license, works must always be attributed to the copyright holder (original author), cannot be used for any commercial purposes, and may not be altered. Any other use would require the permission of the copyright holder. Students may inquire about withdrawing their dissertation and/or thesis from this database. For additional inquiries, please contact the repository administrator via email (scholarship@uwindsor.ca) or by telephone at 519-253-3000 ext. 3208.
Fast Modular Reduction for Large-Integer Multiplication

by

Suhas Sreehari

A Thesis
Submitted to the Faculty of Graduate Studies
through Electrical and Computer Engineering
in Partial Fulfillment of the Requirements for
the Degree of Master of Applied Science at the
University of Windsor

Windsor, Ontario, Canada

2012

© 2012 Suhas Sreehari
Fast Modular Reduction for Large-Integer Multiplication

by

Suhas Sreehari

APPROVED BY:

______________________________________________
Dr. A. Reza Riahi
Department of Mechanical, Automotive and Materials Engineering

______________________________________________
Dr. Mohammed A.S. Khalid
Department of Electrical and Computer Engineering

______________________________________________
Dr. Majid Ahmadi, Co-Advisor
Department of Electrical and Computer Engineering

______________________________________________
Dr. Huapeng Wu, Co-Advisor
Department of Electrical and Computer Engineering

______________________________________________
Dr. Govinda Raju, Chair of Defense
Department of Electrical and Computer Engineering

August 2012
DECLARATION OF ORIGINALITY

I hereby certify that I am the sole author of this thesis and that no part of this thesis has been published or submitted for publication.

I certify that, to the best of my knowledge, my thesis does not infringe upon anyone’s copyright nor violate any proprietary rights and that any ideas, techniques, quotations, or any other material from the work of other people included in my thesis, published or otherwise, are fully acknowledged in accordance with the standard referencing practices. Furthermore, to the extent that I have included copyrighted material that surpasses the bounds of fair dealing within the meaning of the Canada Copyright Act, I certify that I have obtained a written permission from the copyright owner(s) to include such material(s) in my thesis and have included copies of such copyright clearances to my appendix.

I declare that this is a true copy of my thesis, including any final revisions, as approved by my thesis committee and the Graduate Studies office, and that this thesis has not been submitted for a higher degree to any other University or Institution.
ABSTRACT

The work contained in this thesis is a representation of the successful attempt to speed-up the modular reduction as an independent step of modular multiplication, which is the central operation in public-key cryptosystems. Based on the properties of Mersenne and Quasi-Mersenne primes, four distinct sets of moduli have been described, which are responsible for converting the single-precision multiplication prevalent in many of today's techniques into an addition operation and a few simple shift operations. A novel algorithm has been proposed for modular folding. With the backing of the special moduli sets, the proposed algorithm is shown to outperform (speed-wise) the Modified Barrett algorithm by 80% for operands of length 700 bits, the least speed-up being around 70% for smaller operands, in the range of around 100 bits.
To my family – for the consistent love, support, and faith.
ACKNOWLEDGEMENTS

“No stream or gas drives anything until it is confined. No Niagara is ever turned into light and power until it is tunneled. No effort ever grows great until it is focused, channeled, dedicated and disciplined.” This is what one of my undergraduate professors always says.

I realized that first hand during the two years I spent at the University of Windsor. I still remember how I was at the beginning of my MASc program. I was excited, but clueless. I had studied several courses during my undergraduate years too, but it was the first time I was about to conduct research. It meant I had to find out something, do something, however tiny – that no one else had done before. Not just that, whatever I do has to be useful. That is where my supervisors came into picture.

Dr. Huapeng Wu (my supervisor) taught me a lot about feasibility. He showed me the importance of always staying on top of literature, and made sure my research did not become obsolete. Dr. Majid Ahmadi (my co-supervisor) spent a lot of time motivating me, and always encouraged me to try new things. Without the constant support and supervision of these two professors, I would not have put together the work contained in this thesis. I also would like to thank Dr. Reza Riahi (my external program reader) and Dr. Mohammed Khalid (my department reader) for their time and constructive suggestions.
Payal Khanwani deserves my heartfelt gratitude for all her support and love. Ms. Andria Ballo (ECE graduate secretary) and Ms. Rachelle Marchand (ECE undergraduate secretary) have been thoroughly patient (I have run to them all the time for every small thing) and have indirectly helped me graduate with success.

Finally, I would like to mention the continuous support from my family, and their faith in me – which helped me focus just on my research.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>DECLARATION OF ORIGINALITY</td>
<td>iii</td>
</tr>
<tr>
<td>ABSTRACT</td>
<td>iv</td>
</tr>
<tr>
<td>DEDICATION</td>
<td>v</td>
</tr>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>vi</td>
</tr>
<tr>
<td>LIST OF FIGURES</td>
<td>x</td>
</tr>
<tr>
<td><strong>Chapter I: Introduction</strong></td>
<td>1</td>
</tr>
<tr>
<td>1.1 Defining Large</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Motivation</td>
<td>2</td>
</tr>
<tr>
<td>1.3 The big picture</td>
<td>4</td>
</tr>
<tr>
<td>1.4 Organization of the thesis</td>
<td>5</td>
</tr>
<tr>
<td><strong>Chapter II: Preliminaries</strong></td>
<td>7</td>
</tr>
<tr>
<td>2.1 Recapitulating abstract algebra</td>
<td>7</td>
</tr>
<tr>
<td>2.2 Finite fields and arithmetic</td>
<td>14</td>
</tr>
<tr>
<td>2.3 Prime numbers</td>
<td>16</td>
</tr>
<tr>
<td>2.4 Public-key cryptography</td>
<td>18</td>
</tr>
<tr>
<td>2.5 RSA cryptography</td>
<td>21</td>
</tr>
<tr>
<td><strong>Chapter III: Historical overview and technical survey</strong></td>
<td>25</td>
</tr>
<tr>
<td>3.1 Large-integer multiplication</td>
<td>25</td>
</tr>
<tr>
<td>3.2 Modular multiplication/reduction techniques</td>
<td>38</td>
</tr>
<tr>
<td><strong>Chapter IV: Proposed method and results</strong></td>
<td>50</td>
</tr>
<tr>
<td>4.1 Methodology</td>
<td>50</td>
</tr>
</tbody>
</table>
4.2 Results

Chapter V: Conclusion and future work

5.1 A summarized view of the thesis

5.2 Scope for future work

REFERENCES

VITA AUCTORIS
LIST OF FIGURES

Figure 1: The Big Picture 4

Figure 2: A simplified illustration of public-key cryptography (for data security) 21

Figure 3: A brief timeline of the some important works pertaining to modular reduction geared towards multiplication 40

Figure 4: Time delay comparison 62

Figure 5: Time delay comparison – smoothed graph 62

Figure 6: Relative speed-up (as a percentage) 63
“Lots of people working in cryptography have no deep concern with real application issues. They are trying to discover things clever enough to write papers about.”

- Whitfield Diffie (of the Diffie-Hellman Key Exchange fame)

I hope and believe my work presented in this thesis does not put me with those “lots of people”.

1.1 Defining “large”

Arithmetic operations are the often thought of to be the most understood concepts. While this seems to be true for smaller bit-lengths, the situation gets trickier in that the school-book method that works so well in implementing smaller operand-arithmetic suddenly seems slower and cumbersome in implementing the arithmetic for large operands. Now, “large” is a subjective qualifier, and it is hard to put a number on it in generic terms. What is large for one application could be not-so-large or even small for another. So it is important that we outline the application we are targeting the development of the current arithmetic towards. The operation in question throughout this thesis is the multiplier-based modular reduction of a large integer. Application-wise, this would be a typical step in modular exponentiation – which is routinely carried out in RSA cryptography. In this light, “large” integers typically are 200-500 bits in length at the core, with a 50-bit tail at
the lower end and up to a 200-bit tail at the upper end making the extended range much wider, from 150 to 700 bits in length. However, in this thesis, when not explicitly mentioned, a safe assumption is that a large integer is around 500 bits long, on an average.

Now that there is some clarity as to what constitutes a large integer for our purposes, let us see where the problem tackled in this thesis fits (in the big picture) and more importantly, what the problem I have solved is.

1.2 Motivation

Very simply put, the problem this thesis sets out to solve, and subsequently solved, is to speed up the computation of the product of two large integers modulo a prime modulus half the bit-length of the product. Now, there exist many solutions (some of which are neat and elegant) to multiply two large integers – whatever way, be it bit-serially, or word-serially, or in parallel, depending on the time, area, and power considerations and based on the available restrictions and resources.

Then we have algorithms which multiply and modulo-reduce integers alternatively, taking small chunks of the operands, in a sliding windowed fashion. This method is generally word-serial.

There are very famous algorithms to reduce large integers. It came to my notice early on that the amount of work gone into bettering these reduction methods is not much at all,
compared to amount of time and effort spent by the research community on improving multiplier designs.

After recognizing these three broad categories, it was decided to investigate the merits and demerits of all three approaches before adopting one for this thesis work. Not to put too fine a point upon it, let us see the high-level factors which helped in deciding which stream to choose.

Based on [30], the word-serial multiply-reduce alternation scheme was ruled out because there is almost no speed gain in this method compared to first multiplying the large integers, and following it up with reduction. However, in separating multiplication and reduction, the freedom gained in terms of using algorithms independently of each other (for multiplication and reduction) – as long as there is basic I/O compatibility – is enormous. This shines the spotlight on three possibilities – (i) working on the multiplier unit, or (ii) working on the reduction unit, or (iii) working on both. Time restrictions on master’s degree took care of eliminating the third option. As for working on the multiplier, there exist many innovative algorithms and seems to be the general focus of researchers. The scope for improvement is bigger in the reduction unit camp, and it was felt that this research area is somewhat neglected, if not abandoned. These realizations provided enough clarity and motivation to work mainly on the reduction unit, while adopting the best multiplier unit to go with it, to build a complete modular multiplier block.
1.3 The big picture

To remain sufficiently motivated and clear about the problem (and the corresponding solution proposed in this thesis), it is essential we resort to *the big picture*.

As we can see in Fig. 1, Modular Reduction forms an integral part of Modular Multiplication, which in turn is an important step in carrying out Modular Exponentiation operations. Modular Exponentiation operations are the backbone of any RSA cryptosystem implementation, but are very expensive in terms of hardware complexity. Therefore, speeding up Modular Reduction at the lowest level has far reaching consequences all the way up to Modular Exponentiation, and thus the entire cryptosystem itself.
1.4 Organization of the thesis

The thesis is organized in such a way to retain maximum clarity and focus on the problem of central importance. As we have already seen, Chapter 1 (the introductory chapter) has been kept relatively succinct to the point of highlighting the motivation and academic justification for choosing the problem this thesis has indeed succeeded in bringing about the desired solution for.

Chapter 2 is solely dedicated to rapidly bring any novice reader up to the technical comfort, that is required to not only grasp the implications of the work and results presented in this thesis, but also to fully appreciate and add to the existing richness by way of constructive critiques and/or augmenting the work as suggested in the last chapter, under the section which delineates possibilities for future work. To achieve this clear-cut objective, crisp explanations of certain well-known and required mathematical models, slowly built up from more accessible mathematical foundations, have been included. As a result, any scholar in this area might find it tempting (and justifiably so) to skip this chapter.

Chapter 3 makes best use of the knowledge built up in Chapter 2, by introducing the state-of-the-art work in this area, in a phased manner. Now, what is meant by a phased manner is that the actual state-of-the-art introduction is preceded by a succinct historical
overview of the evolution of this research area, and is seamlessly merged into the most modern work, in a blurred boundary fashion.

Chapter 4 is a portrayal of the original work and contribution of this thesis to extending the envelope of the state-of-the-art technique in speeding up Modular Reduction, in a backwards compatibility mode with the most widely used classic multiplication algorithms. The theoretical formulation of the solution is immediately backed by FPGA implementation results of the proposed algorithm, and the chapter concludes with the concurrence of the expected result (from the theoretical side) and the achieved results (from the implementation side), thus validating the claims made in this thesis.

Chapter 5 is a rather short, but tight stitch-up of the myriad ideas scattered throughout the thesis, so as to form one self-contained block which encompasses the problem, the deficiencies of existing solutions, my proposed solution, why my solution works, and finally scoping the possibilities for future work.
In this portion of the thesis, a narrow down strategy is adopted to get to the problem definition. Actually, it is only in the next section that the problem would be defined in explicit terms, but the role of the current section is to provide the reader with ample mathematical and conceptual background in order to better understand the future sections lined up.

To that end, the basics of abstract algebra would be covered, brushing along group theory, and diving into finite fields and their arithmetic. Once we are done with the special arithmetic defined within finite fields, the platform would be set for a general introduction to the working of the public-key cryptographical technique outlined by Rivest, Shamir, and Adleman, now widely known as “RSA cryptography”.

2.1 Recapitulating abstract algebra

The study of algebraic structures like groups, fields, rings, and vector spaces fall into the purview of “abstract algebra”. The term abstract algebra came into being at the beginning of the 20th century and it helped distinguish this area from elementary algebra which deals with the algebraic expressions, their solutions, with real and imaginary components.
Major themes handled in abstract algebra range from solution of systems of linear equations, which lead to linear algebra to closed form expressions representing the solutions of general polynomial equations of higher degree that resulted in discovery of groups as abstract manifestations of symmetry. Beyond this, arithmetical investigations of quadratic and higher degree forms and Diophantine equations have been carried out in this area of mathematics. There are several problems that figure in the grasp of abstract algebra, but we will stop here as the point of how important and influential this branch of modern mathematics is, is made well.

Now, we shall begin the exploration from basic set theory, and then move on to groups.

### 2.1.1 Sets

A “set” simply is a collection of well-defined entities/objects, which are connected together by some common thread – which would serve as the characteristic of the set. The members of a set are called *elements* of the set. Basic examples of sets could be, 

W = {Violet, Indigo, Blue, Green, Yellow, Orange, Red}, where W is the set of all the colours in white light.  

P = {2, 3, 5, 7}, where P is the set of all prime numbers between 0 and 9.
In a lot of cases, enumeration of the elements of a set is often impractical, and tedious. Some sets are even infinite, in which case enumeration is not even possible. In all those situations, stating the rule which governs the set is the best way to describe the set.

Suppose E is an element of set A. Then, it is represented by $E \in A$ (read “E belongs to A”). However, if an element F (which is an element of another set C) is not an element of set A, then we write $F \not\in A$ (read “F does not belong to A”).

It is possible that one could identify one set as being fully a part of another. In set theory language, this is the concept of “sub-sets”. If A and B are two sets, and if A is fully encompassed in B, then we say that A is a subset of B, denoted by $A \subseteq B$. This means that all the elements of set A are also elements of set B. Now, if set B has elements that are not found in set A, A is called a proper subset of B, denoted by $A \subset B$. A very simple example would be,

A = \{2, 3, 5, 7\}, where A is the set of prime numbers between 0 and 9.

B = \{2, 3, 5, 7, 11, 13, 17\}, where B is the set of prime numbers between 0 and 19.

It is fairly easy to see that the elements of A are completely contained in set B, and also set B has more elements that set A does not. In other words, $A \subset B$.

Further, the number of elements in any set is called the “cardinality” of the set, or simply the cardinal number. The reader may wish to recall the operations that are usually defined over sets, such as unions, intersections, Cartesian products, and inversions (generally known as complementation).
It is time to move on to the topic of primary importance in modern algebra – “algebraic structures”. We come across various types of structures, and let us see what those are.

2.1.2 Algebraic structures

An algebraic structure contains multiple sets, closed under certain operations. Groups, fields, and rings are all structures. Broadly, structures are divided into two kinds – those whose axioms are identities, and those in which some axioms may not be identities. Group-like structures belong to the former category, while field-like structures belong to the latter category. We will touch upon groups and fields later. All the axioms mentioned hereunder are taken from Peter Cameron’s course notes on algebraic structures [1].

2.1.3 Groups

A group is an algebraic structure with just one binary operation, and it satisfies four axioms:

(G0) (Closure law) For any g, h ∈ G, we have g * h ∈ G.

(G1) (Associative law) For any g, h, k ∈ G, we have (g * h) * k = g * (h * k).

(G2) (Identity law) There is an element e ∈ G with the property that g * e = e * g = g for all g ∈ G. (The element e is called the identity element of G.)

(G3) (Inverse law) For any element g ∈ G, there is an element h ∈ G satisfying g * h = h * g = e. (We denote this element h by g⁻¹, and call it the inverse of g.)
Additionally, if the group $G$ satisfies the following law, then it is termed an “Abelian Group”:

\[(G4)\text{ (Commutative law)}\text{ For any } g,h \in G, \text{ we have } g \ast h = h \ast g.\]

2.1.4 Rings

A ring is defined by two operations: addition (+) and multiplication (.). Sometimes, we ignore the “.” In the multiplication, and simply concatenate the operands. We define a ring to be a set $R$ with two binary operations satisfying the following axioms.

Axioms for addition:

\[(A0)\text{ (Closure law)}\text{ For any } a,b \in R, \text{ we have } a+b \in R.\]
\[(A1)\text{ (Associative law)}\text{ For any } a,b,c \in R, \text{ we have } (a+b)+c = a+(b+c).\]
\[(A2)\text{ (Identity law)}\text{ There is an element } 0 \in R \text{ with the property that } a+0 = 0+a = a \text{ for all } a \in R\. \text{ (The element } 0 \text{ is called the identity element of } R).\]
\[(A3)\text{ (Inverse law)}\text{ For any element } a \in R, \text{ there is an element } b \in R \text{ satisfying } a+b = b+a = 0. \text{ (This element is } -a, \text{ and we call it the additive inverse of } a).\]

On similar lines, we have the axioms for multiplication:
(M0) (Closure law) For any $a,b \in R$, we have $ab \in R$.

(M1) (Associative law) For any $a,b,c \in R$, we have $(ab)c = a(bc)$

Apart from those, we do have a mixed axiom as well:

(D) (Distributive laws) For any $a,b,c \in R$, we have $(a+b)c = ac+bc$ and $c(a+b) = ca+cb$.

These are the basic axioms in place. However, there are further multiplicative properties listed below:

(M2) (Identity law) There is an element $1 \in R$ such that $a1 = 1a = a$, for all $a \in R$. (The element $1$ is called the identity element of $R$.)

(M3) (Inverse law) For any $a \in R$, if $a=0$, then there exists an element $b \in R$ such that $ab = ba = 1$. (We denote this element $b$ by $a^{-1}$, and call it the multiplicative inverse of $a$.)

(M4) (Commutative law) For all $a,b \in R$, we have $ab = ba$.

A ring which satisfies (M2) is termed a ring with identity; a ring which satisfies (M2) and (M3) is termed a division ring; and a ring which satisfies (M4) is termed a commutative ring.

A ring which satisfies all (M2), (M3), and (M4), is called a “field”. A ring can have sub-rings too. A sub-ring is a subset of a ring, which by itself is a ring.
2.1.5 Fields

In modern algebra, fields are a class of rings. Perhaps the simplest definition of a field is that it is a commutative ring, the non-zero elements of which form a group under multiplication. What this immediately means is that a ring has a looser existence than a field. A field is expected to have more than one element, thus ruling out the trivial solution that comes from a zero element ring. The specific advantage of a field comes in the form of division by non-zero elements, which is not allowed in a ring.

A field has a useful feature of being capable as a scalar for a vector space, which is the standard general context for linear algebra. The theory of field extensions (which includes Galois theory) involves polynomial roots with coefficients in a field. In modern mathematics, number theory is greatly benefitted by field theory. There are several other areas within mathematics depending on fields, rendering the theory of fields extremely important.

The easiest way to check if a set F is a field is by checking for the following:

1. Closure of F under addition and multiplication:
   For any a, b ∈ F, a+b ∈ F and ab ∈ F.

2. Associativity under addition and multiplication:
   For any a, b, c ∈ F, we have, a+(b+c) = (a+b)+c and a(bc) = (ab)c.

3. Commutativity under addition and multiplication:
   For any a, b ∈ F, we have, a+b = b+a and ab = ba.

4. Additive and multiplicative identity elements:
There exists an element, $0 \in F$, such that for any $a \in F$, $a + 0 = a$. “0” is called the \textit{additive identity element} of $F$. On similar lines, there exists an element, $1 \in F$, such that for any $a$ in $F$, $a \cdot 1 = a$. “1” is called the \textit{multiplicative identity} element of $F$. In order to exclude the trivial ring, the additive identity and the multiplicative identity are required to be distinct.

5. \textbf{Additive and multiplicative inverse elements:}

For any $a \in F$, there exists an element $-a \in F$, such that $a + (-a) = 0 \in F$. Similarly, for any non-zero $a \in F$, there exists an element $a^{-1} \in F$, such that $a a^{-1} = 1 \in F$.

The existence of additive and multiplicative inverse elements allow for subtraction and division within the field, respectively.

6. \textbf{Distributive property:}

For any $a, b, c \in F$, the following equality holds: $a(b+c) = (ab)+(ac)$.

$F$ is Abelian under addition, and $F\setminus\{0\}$ is Abelian under multiplication.

\textbf{2.2 Finite fields and arithmetic}

Describing a finite field could not get easier than stating that finite field is a field with finite cardinality.

The order of a finite field is by definition the same as the number of elements it has, or simply the cardinality. If $q$ is the order of the finite field in question, then it is to be noted that there is a hard constraint on $q$, in that $q$ must be a prime power. In other words, $q = p^x$, where $p$ is a prime number, and $x$ is a positive integer. Typically, $p$ is referred to as the “characteristic of the finite field”. In the case where $x = 1$, $F$ is termed a “prime field”. However, if $x > 1$, $F$ is termed an “extension field”. There is essentially just a single finite field of order $q$. In popular tradition, all finite fields of order $q$ are structurally the same,
and are termed “isomorphic”, represented either by F<sub>q</sub> or GF(q). This thesis would stick with the latter notation throughout. GF stands for “Galois Field”.

Let us first look at prime fields. In prime fields, x = 1, and therefore, q = p. That makes GF(q) = GF(p).

In general, GF(p) = \{0, 1, 2, 3, ….., p-2, p-1\}. One important thing to remember in all our dealings with prime finite fields is that the arithmetic followed within the finite fields is modular in nature. The results of all operations in the finite field are reduced to a number between 0 and p-1 (including both 0 and p-1).

An example would be GF(17) = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16\}.

Then we have the extension fields, where x > 1, and therefore, q = p<sup>x</sup>. That makes GF(q) = GF(p<sup>x</sup>).

In general, GF(p<sup>x</sup>) = \{a_0 + a_1n + a_2n^2 + …. + a_{x-1}n^{x-1}\}. Here, \{a_0, a_1, ….., a_{x-1}\} are coefficients, and their values range from 0 to p-1. All the polynomials in an extension field are reduced to below n<sup>x</sup>. The most widely used value of p in an extension field is 2, giving rise to the name, binary extension field. Clearly, in a binary extension field, the coefficients can only take two values, 0 or 1.

An irreducible polynomial f(n) of degree x is chosen (such a polynomial exists for any value of x and can be efficiently found). Irreducibility of f(n) means that f(n) cannot be factored as a product of binary polynomials each of degree less than x. Addition of field elements is the usual addition of polynomials, with coefficient arithmetic performed modulo p. Multiplication of field elements is performed modulo the reduction polynomial
f(n). For any binary polynomial \(a(n)\), \(a(n) \mod f(n)\) shall denote the unique remainder polynomial \(r(n)\) of degree less than \(x\) obtained upon long division of \(a(n)\) by \(f(n)\); this operation is called reduction modulo \(f(n)\).

An example of a binary extension field would be \(GF(2^5) = \{a_0 + a_1n + a_2n^2 + a_3n^3 + a_4n^4\}\).
Here \(a_0, a_1, a_2, a_3,\) and \(a_4\) are all binary numbers, meaning they can be 0 or 1 only.

### 2.3 Prime numbers

It is also important that we briefly understand certain types of prime numbers to fully appreciate the formulation of the proposed methodology, which is set to be unfolded later.

In their most basic, yet sufficient, definition, prime numbers are those integers (bigger than unity) whose integral divisors are unity (one) and the number itself. However, even with this simple definition, there does not exist a formula for generating prime numbers, and a number when claimed to be prime has only to be verified by ruling out all other factor-candidates other than the trivial ones indicated above, i.e., one and the number itself.

That said, there are several types of prime numbers – based on how they can be expressed, and on what form they appear. There are quite a few useful and intriguing theorems and hypotheses (the most famous being the Riemann hypothesis) regarding the behavior, distribution, and density of prime numbers.
Let us look into three types of prime numbers, which will surface again during formulation of the proposed methodology.

(1) Proth Primes: It is a prime number which can be expressed as \( P_{proth} = \left( (2h + 1) \cdot 2^n \right) + 1 \). If we denote the odd number generated by \((2h+1)\) by \(q\), then \(q\) has to be bigger than \(2^n\), \(n\) being a positive integer. Divisors of Fermat numbers satisfy this condition, unless \(h\) is negative or non-integer – with a magnitude of \((2^n - 1)/2\) or bigger.

Proth primes satisfy what is known as the Proth's theorem, i.e., “a number \(N\) of this form is prime iff there exists a number \(a\) such that \(a^{(N-1)/2}\) is congruent to -1 modulo \(N\). This provides an easy computational test for Proth primes. Yves Gallot has written a downloadable program for testing Proth primes and many of the largest currently known primes have been found with this program.” [2]

(2) Solinas Primes: It is a slight deviation from the to-be-introduced Mersenne primes. It is named after Jerome Solinas, and takes the form \(2^x \pm 2^y \pm 1\). It is essential that both \(x\) and \(y\) be positive integers, and that \(x\) be bigger than \(y\).

(3) Mersenne Primes: These are perhaps more famous than the two types above. Mersenne numbers are of the form \(2^x - 1\). Now, these Mersenne numbers, because of their convenient unit-shortage of a power of two, can be especially useful in any mathematical operation which is trivially inexpensive on a power of two; the results on the power of two can be typically extended by a simple addition or an equally inexpensive operation to the Mersenne number.

The necessary condition for the Mersenne number to be prime is that \(x\) has to be prime. Caution should be exercised in noting that \(x\) being prime does not guarantee the corresponding Mersenne number being prime. In that sense, the condition is necessary, but not sufficient. From the other viewpoint, \(x\) being composite is a sufficient (but not necessary) condition for the corresponding Mersenne number to be composite.
2.4 Public-key cryptography

To understand the role modular multiplication plays in modern day cryptography, it is imperative to understand how cryptography is operated. The two main forms are the “symmetric-key cryptography” and the “public-key cryptography”.

Public-key cryptography is a secure system that operates differently from traditional cryptosystems in the requirements of the keys. While traditional methods (like the popular Caesar cipher) operate with just one key – for both encryption and corresponding decryption, public-key cryptography demands the usage of two separate keys – the first one to encrypt the plaintext, and second to decrypt the cipher-text. Therefore, the keys no longer have the same function. Now, the genius of this type of key usage lies in fact that only one of these keys is made public, while the other is maintained in full secrecy. The former is called the “public key” – thus giving name to the system of cryptography, and the latter is termed the “private key”. These keys actually form a dual-purpose system. This will be discussed later. For now the only purpose we are concerned with is the data security, for which the public key is the one used for encryption and the private key for decryption.

The above methodology employs asymmetry in the way these keys are put to use. This is a stark contrast from symmetric-key cryptography which as mentioned earlier uses a single key algorithm to provide data protection. The widely perceived advantage of the public-key methodology is that a potential hacker virtually cannot derive the private key
from the knowledge of the public key. So, what make the system hard to crack are just not the keys, but the intricate, extremely convoluted relationship between them – causing most efforts to hack futile within reasonable limits of time.

The public key is advertised, so to say, and it is the job of the sender to use this public key of the recipient to “wrap up” his/her message in a form that is “un-wrappable” only by the intended recipient. By making the encryption key public, the key publisher enables anyone to send him/her messages securely. The only pre-requisite is that the sender should have the public key of the receiver, but it is never a problem since the receiver public key is openly available to everyone. When anyone desires to send a secret message to a particular person, the sender encrypts it using the intended recipient’s public key. At the receiver’s end, the corresponding private key is made use of in order to decrypt the secret message. This way it is secure. In fact it is so secure that the sender himself/herself cannot break his/her own encrypted message!

Hence, public-key cryptography successfully eliminates completely the need for the initial exchange of the key in a secure manner. The asymmetry applies also to the difficulty involved in hacking and genuinely receiving the messages. The way these types of crypto-systems work is that the receiver can easily set up his system. By system, we mean the configuration of the public and private keys. The relationship between the private and public keys however is largely convoluted, making it extremely difficult for anyone to figure out the private key based on their knowledge of the public key. The relationship between the public key and the private key is mathematical, and guessing (or working out the details of the private key with the knowledge of the public key) usually involves integer factorization and discrete logarithms. These problems are not known to
have any efficient solutions within polynomial time. It is this feature that makes public-key crypto-systems next to impossible to hack.

However, with all these features in place, security attack is still a threat, forcing the usage of increasingly larger keys.

We still need to visit the other possibility afforded by these public-key systems: the case where the role of the private key and the public key are interchanged. This finds application to provide authentication – to determine that the message has indeed been sent by the intended sender.

This technique is called “Digital Signature”. The concept is simple in that the sender encrypts the message with his/her own private key. This message is not secure, since anyone can un-wrap the message using the sender’s public key (which is easily available).

The point of digital signatures is not data security, it is merely authentication. The authentication is made possible because when a certain message in encrypted by the sender’s private key, only the corresponding public key can decrypt the message.

It is also possible to have both data security and authentication in the same system.
These are just the basics of how public-key cryptography works. To understand where the proposed work fits in, one needs to go into the workings of the RSA cryptography specifically. It does not mean to limit the application of the proposed work (or any that figure in the technical survey) to just RSA cryptography. The applications go beyond RSA crypto-systems, but it is just that if anyone were to name one application it would probably be RSA crypto-systems.

2.5 RSA cryptography

Before beginning the description of RSA encryption, it would be helpful to review three well-known theorems: Fermat’s Little theorem, Fermat’s Extended Theorem, and the Chinese Remainder theorem.
A. Fermat’s Little Theorem

Let p be a prime number, and a be an integer co-prime with p. This can happen if a is not an integral multiple of p. That is, GCD(a, p) = 1. Note that GCD is the Greatest Common Divisor. Then,

\[ a^{p-1} \equiv 1 \mod p. \]

B. Fermat’s Theorem Extension

If GCD(p,q) = 1, then \( p^{\phi(q)} \equiv 1 \mod q \), where \( \phi(q) \) represents the number of integers less than m that are co-prime with m – which is essentially the Euler-Totient function. The number m is not necessarily prime.

C. Chinese Remainder Theorem

Let p and q be two numbers (not necessarily primes), but such that GCD(p,q) = 1. Then if \( a \equiv b \mod p \) and \( a \equiv b \mod q \), we have,

\[ a \equiv b \mod pq. \]

The RSA algorithm

The main intention of the RSA algorithm is to encrypt the message so as to keep it from everybody except the one with the proper key to decrypt the message. Now, this forms the core idea of any cryptosystem is the same. Unlike steganography where the very existence of the message is hid, all cryptographic techniques rely on strong scrambling of the message. The challenge lies in scrambling the message in such a way that it is impossible to hack the message (by way of brute force trial-and-error or guess work, or
by way of working out the decryption key systematically) within the useful confines of
time and resources.

Rivest, Shamir, and Adleman (RSA being the most common abbreviation for the trio) put
forth an algorithm for carrying out cryptography asymmetrically. In other words, the
RSA algorithm was a public-key technique, and therefore the algorithm does not include
key-sharing.

There are two main stages in carrying out RSA cryptography. The primary part deals
with the generation of private and public keys, followed by the part that deals with the
actual encryption and decryption.

**Algorithm for generating the keys:**

Step 0: Come up with two large prime numbers (say p1 and p2), not too disparate in size.
Step 1: Let n = pq, and it follows that the Euler-Totient function, \( \phi = (p - 1)(q - 1) \).
Step 2: Choose randomly an integer \( e \) (1 < e < \( \phi \)), such that \( \text{GCD}(e,\phi) = 1 \). [e is co-prime
with \( \phi \).]
Step 3: Compute an integer \( d \) (1 < d < \( \phi \)) using the Extended Euclidean algorithm, such
that \( d \) is the inverse of \( e \) modulo \( \phi \). In other words, \( ed = 1 \mod \phi \).

The public key is, then, \( \{n,e\} \), while \( d \) is the private key (which is kept secret).
Next, let us look at the algorithm which specifically represents the encryption and decryption parts.

*Algorithm for encryption and decryption:*

A. Encryption  
Premise: Alice is the sender, Bob is the recipient.

Step 1: Alice obtains Bob’s public key, \{n,e\}. 
Step 2: The message \( m \) is formulated, such that \( 0 \leq m < n \). 
Step 3: Send \( c \) (ciphertext) = \( m^e \mod n \), to Bob.

B. Decryption

Step 1: Bob receives the ciphertext, \( c \). 
Step 2: Reconstruction of the message, \( m = c^d \mod n \). (This is possible only with Bob’s private key, \( d \).)
3.1 Large-integer multiplication

It is considered best to start this survey of literature in the area of large integer multiplication via two parts. In the first (which is essentially section 3.1.2), an attempt is made to make a compilation of the important multiplier designs and multiplication algorithms chronologically. In the second part (section 3.1.3), an outline of the papers and patents that make an impact on this research is presented, again chronologically.

Apart from this chronological placement and the sectioning, this thesis introduces a “Match Grade” for each piece of work that is referred to in this survey chapter. Grade A indicates a very close resemblance to this research topic and/or a high degree of impact it bears upon the proposed work. Grade B is work that has been carried out in this area, and is very relevant to the proposed work. Grade C is for the foundational work that has been done leading into this area of research, and is distinctly different form central thesis problem definition.

Finally, after covering both sections of the literature survey, this thesis specifies the proposed research statement.
Before beginning with the aforementioned two sections, completeness demands an introduction of the topic from a distance – so that we better appreciate the scope of the work and the motivations, for it is very easy to lose track of the broader picture as we delve into the finer contributions of the many articles showcased in this report. Let us begin with a bird's eye view of multiplication, and swoop in to the problem statement by the end of this report.

3.1.1 Multiplication and Basic Multipliers

Multiplication at the most basic level is simply accumulating a number upon zero for a given number of times. The cost of realizing this innocuous looking operation cannot be underestimated.

Let us define a cost function $C(n)$ which estimates the number of smaller multiplication and addition operations needed to accomplish the bigger multiplication. Without loss of generality, and out of customary practice, we can consider that both operands of the bigger multiplication are both of the same length of $n$ bits. To put the research activity in this area succinctly, all efforts to optimize multiplication has centered around reducing the degree and the coefficients of this function, $C(n)$.

The elementary school method of carrying out multiplication may be the easiest to remember, but the worst to implement – due to its high cost.
As an example, consider, $A = a_1x + a_0$, and $B = b_1x + b_0$. $C = A*B = (a_1*b_1)x^2 + (a_1*b_0 + b_1*a_0)x + a_0*b_0$. Expanding the original multiplication this way is indeed the elementary school method. This method requires 4 smaller multiplication operations, and 3 addition operations. In general, this method requires $n^2$ multiplications, and $(n-1)^2$ additions. This can be thought of to be the upper limit of $C(n)$ of all multiplication algorithms.

Since all coefficients are binary in nature, these multiplication operations are realized by way of building digital circuits. Therefore, reducing the number of gates directly decreases the chip area, while parallelizing the structure decreases the time delay. There are many hardware multipliers based on popular algorithms (like the Booth algorithms, BMK method, Wallace tree approach, and so on), but these work economically only for small operands. Let us look at some algorithms that are better suited for large integer multiplication.

### 3.1.2 Fast multiplication algorithms for large integer operands

#### 3.1.2.1 Karatsuba Algorithm

[Match Grade: B+]

Put forth by Anatolii Karatsuba [3] in 1962, the basic principle of this algorithm is to reduce the number of single-digit multiplications needed to achieve an $n \times n$ multiplication – from $n^2$ to $m$, where $m$ is at most $3n \log_2 3$. For the specific case where $n$ is a power of 2, $m$ reduces to $n \log_2 3$. Hence the Karatsuba algorithm has an asymptotic complexity of $\Theta(n^{1.585})$. 

Consider two numbers of n-digits represented in some base b (which typically is 2), x and y. Then we can represent them as a sum of smaller segments:

\[ x = x_1 b^p + x_0 \]
\[ y = y_1 b^p + y_0 \]

It is to be noted that \( p \) is less than \( n \), and \( x_0 \) and \( y_0 \) are each lesser than \( b^p \).

Then,

\[ xy = b^{2p}(x_1 y_1) + b^p(x_1 y_0 + x_0 y_1) + (x_0 y_0) \]

We can replace \( xy \) by \( z \), \( x_1 y_1 \) by \( z_2 \), \( (x_1 y_0 + x_0 y_1) \) by \( z_1 \), and \( x_0 y_0 \) by \( z_0 \). Karatsuba replaced the two multiplications involved in the computation of \( z_1 \) by a single multiplication operation and just a few more addition operations (which are way cheaper than multiplication).

Retaining \( z_2 \) and \( z_0 \) as they are, \( z_1 = (x_1 + x_0)(y_1 + y_0) - z_2 - z_0 \). By doing this, Karatsuba successfully eliminated one multiplication operation. Though this algorithms works for any value of \( n \) and \( p \), it is to be noted that the value of \( m \) hits the minimum when \( n = 2^a \) and \( p = n/2 \) (where \( a \) is a positive integer).

The Karatsuba algorithm is not very useful when dealing with operands of length smaller than 128 bits [4]. This is a rough number and it depends on the platform being used for implementation.
3.1.2.2 Toom-Cook Algorithm

[Match Grade: B+]

Now, let us move on to the more general Tom-Cook algorithm (whose special case is the Karatsuba algorithm) [3]. Introduced first by Andrei Toom in 1963, and subsequently improved by Stephen Cook in 1966, this algorithm involves splitting the operands into $k$ smaller numbers of length $p$ each. As we can readily see, Karatsuba algorithm is the case where $k = 2$. $k = 3$ is also a very famous case (generally referred to as Toom-3), and it operates at a complexity of $\Theta(n^{1.465})$. At this point, it is also good to observe that the elementary school method that I have mentioned in section 2 is also a case of Toom-Cook algorithm with $k = 1$. And hence, it is sometimes known as Toom-1 as well, and it operates at $\Theta(n^2)$. Since we shall be seeing a much better algorithm next, the implementation details of this algorithm will not be discussed.

3.1.2.3 Schönhage-Strassen Algorithm

[Match Grade: C]

This is not a new algorithm. But surely, this has been one of the most powerful algorithm since 1971, when it was introduced by Arnold Schönhage and Volker Strassen [5]. The arithmetic complexity with which it operates is $\Theta(n \log n)$. 
This algorithms recursively calls the FFT in algebraic rings of size $2^{2^n} + 1$. Without going very deep, let us see a quick overview of the working and advantage/disadvantage of the Schonhage-Strassen algorithm.

Consider two numbers A and B to be multiplied. Let A and B have n digits each. Then, their cyclic convolution will also have n entries (although every entry need not be a digit anymore). If carrying is performed leftward from the LSB, then we arrive at the modular product of A and B.

So what we get is $P = A \times B \mod (b^n - 1)$. However, if we route the procedure via negacyclic convolution, we get $P = A \times B \mod (b^n + 1)$. The base $b$ is generally 2, hence this further reduces to,

$P = A \times B \mod (2^n + 1)$. $n$ again is a power of 2, i.e., $n = 2^k$.

The speeding up of the algorithms lies mainly in the employment of Fast Fourier Transform techniques (FFT) to carry out the Discrete Fourier Transform (DFT).

The method is to simply take the operands A and B to the frequency domain by way of applying DFT upon them. Let us denote the frequency domain equivalents of A and B by A' and B' respectively.

Now, A' and B' are multiplied, which we store in $P_1'$. We obtain $P_1$ by applying Inverse DFT (IDFT) to $P_1'$. Again the trick is to use any popular I-FFT technique. FFT and I-FFT are the main speed-up steps.
The primary advantage comes from the speed. But practical Schonhage-Strassen implementations outdo Toom-Cook algorithm only for huge operand sizes (in the range of $2^{2^{15}}$ to $2^{2^{17}}$).

Now, it is time to move on to a survey of papers. We shall start with the early papers that laid a solid foundation to the concept of fast hardware multipliers. These may not necessarily have anything to do with large integer multipliers, but they make way for the development of multipliers. References to them are made chronologically rather than by importance or impact (for the Match Grade takes care of the latter).

3.1.3 Overview of published and presented articles – foundational, relevant, and/or recent

[The Match Grades will be given in the references section.]

3.1.3.1 The 1960s

As early as 1964, C.S. Wallace [6] realized that multiplication is one of the more important operations in the CPU, and he put forth a suggestion to speed up multiplication by viewing multiplication as addition of a number of summands. He argued that reducing the number of such summands, and accelerating the formation and addition of those summands will improve the efficiency of multipliers. Wallace proposed a tree of pseudo-
adders (without any carry-chain propagation) in his paper. He also acknowledges the trade-off between cost and speed as applicable to the multiplier units.

Soon after Wallace's paper, Dadda [7] came up with some schemes for parallel multipliers in 1965. Dadda made use of parallel (n-input, m-output) counters. He proposed a two-step approach: First, two numbers whose sum equals the product of the operands (without carry-chain propagation) are obtained. Then, the product is obtained in a carry propagating adder.

### 3.1.3.2 The 1970s

Following the interest generated in the 1960s to speed up multiplication (having realized its importance), the 1970s saw many new ideas to realize digital multiplication – like parallelization and residue number system.

In 1970, Habibi and Wintz [8] proposed a number of methods for speeding up multiplication. They also compared those methods on the basis of complexity, cost, and speed. Their paper described a parallel multiplier – which employed carry-save technique – for multiplying 10-bit by 12-bit numbers, with a worst case time of 520 ns. Their reported expenditure on the ICs was under $500! The authors also remark that the Dadda-method and the carry-save method use fewer full adders in comparison to the Wallace method.
In 1976, Dadda followed up his own work with fast parallel digital multipliers. [9] Early on in this paper, it is clarified that parallel multipliers come with higher costs and complexity compared to standard multipliers, but in turn parallel multipliers offer speed. We shall discuss no more of this simply due to the fact that Dadda himself states in his paper that the implementation of large parallel counters (needed for large integer operands) is not feasible in terms of the chip size and the delay produced.

In 1977, Soderstrand and Fields published a paper on Residue Number System (RNS henceforth) multiplication (followed by an inherent division by a constant integer – to effectuate multiplication by a fraction) [10]. Except that RNS scheme was used, this paper has more applications to digital filter design than to present-day cryptography.

3.1.3.3 The 1980s

RNS provided the distinct advantage of offering parallelism to circuits because of its innate ability to cut off carry-propagation chains. Thus RNS seemed to have been catching up around that time, as can be seen by the 1981 paper on large moduli multipliers – in which the large moduli multipliers were designed to extend the dynamic range of the moduli (2n-1, 2n, 2n+1) [11].

In 1983, Gnanasekaran demonstrated that an n-bit operand bit-serial input – bit-serial output multiplier can be realized using n 5-input adders [12]. This work is not very useful to my research for two reasons:
(i) The output is bit-serial – starting from the LSB. Output starting from MSB would be more useful.

(ii) The work deals with bit-serial input, whereas my work uses parallel input.

Therefore, Gnanasekaran's approach is a specific case of my intended work.

In the same year, Preparata [13] described a VLSI network to multiply two large integers of the same length – based on the Discrete Fourier Transform. The operation time is $\Theta(\sqrt{n})$ and the chip area is $\Theta(n)$, where $n$ is the length of each operand. We would not go any deeper into this paper, since the mere purpose of mentioning this work is to show that the idea of using DFT for large integer multiplication has been a revisited one (since the famous Schonhage-Strassen Algorithm).

Five years later in 1988, Hsu et al. [12] compared dual, normal and standard bases-based VLSI architectures of finite field multipliers. The purpose is to determine which of the three bases are best from a VLSI implementation viewpoint. To this end, the authors have taken into consideration:

(i) Berlekamp's dual-basis multiplier,
(ii) Massey-Omura normal-basis multiplier, and
(iii) Scott-Tavares-Peppard standard-basis multiplier,

The authors have implemented them with NMOS technology to compare them all. This
paper deals with finite field multiplication and not with regular multiplication, and this thesis believes it provides a good insight into the development of multipliers. The authors concluded that the dual-basis multiplier occupies the smallest chip area, and that the dual-basis multiplier outperforms the other two as the order of the field employed shoots up. They noted that the normal-basis multiplier is more efficient when the task at hand involved computing the inverse of elements, or squaring/exponentiation operations. This comes at the cost of area as the order of the field scales up. The standard-basis multiplier has its own set of advantages too – like the lack of need for basis conversion and the ease of implementation due to the regularity of the structure.

3.1.3.4 The 1990s

In 1990, Hartley and Corbett [13] made note in their paper that digit-serial computations lead to increased efficiency in chip designs. This paper, while not dedicated to multiplication, has a section on digit-serial multiplication – wherein the authors have adopted a parallel array multiplier over the Booth multiplier (attributing this choice to odd operand sizes, and lack of need for the extra speed resulting from usage of the Booth multiplier – since the paper also talked about other operations running in parallel which anyway take longer time). The authors remark that though efficient the Wallace-tree multiplier is not suitable for VLSI implementation. This paper is important in that the authors notice that digit-serial techniques increase efficiency. But other than that, the multiplier choice itself is not great – since the authors have looked at the overall system performance (mainly speed) and not just at stand-alone multiplication speed.
Ashur et al. [14] in 1996 described a systolic digit-serial multiplier in which the initial delay (before the LSB of the output appears) is made independent of the number of bits and the word length. The authors present an architecture wherein the pipelining is reduced to the bit-level. It is shown that by properly controlling the number of pipelining stages and the word length, a desirable compromise can be struck between speed and cost.

The very next year, an article was published [15] that set out to achieve the same result as above: bit-level pipelining – allowing higher speed with reasonably low chip area. However, the focus of this paper is not on achieving higher sample speeds, but instead trading the extra speed for power consumption reduction. They observed that Type-I multipliers [16] fare better in terms of power consumption for larger digit sizes. In 1998, Taiwanese researchers Guo and Wang [17] focused on standard-basis multipliers in their paper on digit-serial systolic multiplier for GF(2^m). The authors claim that their proposed multiplier achieves an output rate of one every \([m/L]\) clock cycles – where L is the digit size. The authors have taken care to keep the maximum propagation delay under check, as L gets large by introducing increased pipelining.

The claim is that pipelining each basic cell further into S+1 stages reduces the maximum delay by roughly S+1. Further, the paper states that digit-serial systolic inverters can be built using \(2^{m-3}\) proposed multiplier units, and that digit-serial systolic exponentiator units can be built with \(2^{m-2}\) of these multiplier units. The mentioned advantage is an improved trade-off between the throughput of the system and the hardware complexity involved.
3.1.3.5 2000 Onwards

Three years after Guo and Wang's publication, in 2001, C.H. Kim et al. [20] came up with a slightly modified version of [17]. It is their claim that their proposed architecture leads to significant reduction in computational delay at the cost of a slight increase in hardware complexity. The authors have used the LSB-first multiplication algorithm described in [21]. The authors further state that the structure of each Processing Element of their proposed multiplier is simpler than Guo and Wang's.

That very year, C. Visavakul and his colleagues at London's Imperial College published their work pertaining to reconfigurable multipliers based on digit-serial structure [22]. In this piece of work, the authors make possible the construction of any 4Mx4N bit reconfigurable multiplier with Flexible Array Blocks (FAB) and digit-serial techniques. They have described the FAB as a 4x4 bit reconfigurable building block made of regular array of adders. The final multiplier is a direct result of a two dimensional cascading of these basic FABs. The authors have modified the original FAB structure presented in [23].

In 2003, M. Hutter et al. [24] published their work on versatile and scalable multipliers – which made use of an efficient degree reducing circuitry interleaved between multiplication stages.
The proposed work of the authors is shown to reduce the critical path of the degree reduction circuit by 1.36 to 3 times (for digit sizes in the range of 4 to 16 bits).

Two years later, in 2005, W. Tang, H. Wu, and M. Ahmadi [25] published their work on VLSI implementation of bit-parallel word-serial finite field multiplier in GF ($2^{233}$). The authors support their decision to use GF ($2^{233}$) (out of the five NIST recommended fields) by stating that it is a large enough field to provide sufficient security to cryptographic applications, while the irreducible trinomial is simple enough to make the multiplication significantly less complex. The ASIC houses both a multiplier unit and a squarer unit. The authors have made use the squarer architecture outlined in [26].

### 3.2 Modular multiplication/reduction techniques

A lot of work has been done in this relatively small area of concentration. However, let us look into only the most important ones. Here’s a timeline of the relevant work:
3.2.1 Montgomery modular multiplication algorithm

[Match Grade: A]

Algorithm:

*Input*: $X_1$ (n bits), $X_2$ (n bits), $P$ (n bits)

*Output*: $A = (X_1 \times X_2) \mod P$

*Step 0*: Choose $R = 2^g$ (for some integer $g$) | GCD(R, P) = 1 and $R > P$
Step 1: Compute residues $\bar{X}_1 = (X_1 \ast R) \mod P$, $\bar{X}_2 = (X_2 \ast R) \mod P$

Step 2: Compute $T = (\bar{X}_1 \ast \bar{X}_2)$; Compute $q = (T \mod R)\ast P' \mod R$

[Note: $\bar{A} = (\bar{X}_1 \ast \bar{X}_2 \ast R^{-1}) \mod P$]

Step 3: $\bar{A} = (T + q \ast P) / R$

Step 4: If $\bar{A} > P-1$, then $\bar{A} = \bar{A} - P$

Step 5: Return $\bar{A}$.

Insight into the working:

Normal modular multiplication involves division by the modulus, in order to bring the product within the modulus. This division often turns out to be more expensive than the original multiplication. Therefore, the most basic rationale behind Montgomery’s algorithm is to avoid that division.

R is chosen to be bigger than P, and the most convenient value that R takes on is an integer power of 2. Let $R = 2^g$. This makes sure that R is co-prime with P. We see that this condition is necessary to bring about the expression,

$RR^{-1} - PP' = 1$

Since we have chosen a value bigger than P for R, $R^{-1} < P$. Also, $P' < R$. Another advantage of having R as a power of 2 is that shifting and bit masking become extremely efficient and simple.

The Montgomery algorithm considers the P-residues instead of the actual operands for the reason that reducing $TR^{-1}$ modulo P does not require division by P (where T is the product of any two P-residues).
Why the Montgomery algorithm works perfectly well the way it does can be seen in the following brief illustration:

Consider two operands, \( x \) and \( y \). The \( M \)-residues of these operands are \( \bar{x} \) and \( \bar{y} \) respectively. These are computed as: \( \bar{x} = xR \mod M \), and \( \bar{y} = yR \mod M \). Let, \( z = xy \mod M \). We can see that \( z \) is our desired output. In the same way as \( \bar{x} \) and \( \bar{y} \) are defined, we can have \( \tilde{z} = zR \mod M \), i.e., \( \tilde{z} = xyR \mod M = (xR)(yR)^{-1} = \bar{x}\bar{y}R^{-1} \mod M \). If we denote the product of \( \bar{x} \) and \( \bar{y} \), with \( T \), then we have, \( \tilde{z} = TR^{-1} \mod M \), and the Montgomery algorithm presents a fast way to compute this expression.

Now, consider the following:

\[
TR^{-1} = TRR^{-1}/R = T(PP' + 1)/R
\]

Let \( d \) be some integer. Then,

\[
((TN' + dR)P + T)/R \mod P = (TP'P + dRP + T)/R \mod P = (TP'P + T)/R \mod P
\]

This means that one could compute \( TP' \mod R \), instead of \( TP' \). Let, \( q = (T \mod R)P' \mod R \).

Let, \( A = (T + qP)/R \)

Now, it is evident that \( A < 2P \), since \( T < RP \) and \( q < R \). Therefore, we need at most one subtraction to bring \( A \) within the modulus \( P \).

So what we have now is the \( P \)-residue of the product.
Cost analysis:

Computation of the P-residues is definitely an overhead that has plagued the otherwise excellent Montgomery algorithm. For the cost analysis here, let us pretend to ignore the residue computation overhead.

Cost to compute q: \((T \mod R)\) comes nearly free (because it is merely bit masking – since R is a power of 2). Then, \((T \mod R)P'\) needs one \(g \times g\) multiplication. This would be the cost to compute q.

Cost to compute A: Calculation of \(qP\) needs one \(g \times 2n\) multiplication. Apart from that, computation of A needs one addition, division by R (which is merely a right shift by \(g\) bits), and possibly a subtraction (which can be viewed as addition complexity-wise).

The value of \(g\) can be conveniently chosen as \(2n+1\).

With this choice of \(g\), the Net Cost: 1 \((2n+1) \times (2n+1)\) multiplication, 1 \((2n+1) \times (2n)\) multiplication, 1-2 additions, and a few bit masks and shifts. This cost excludes the P-residue conversions at the input and output.

### 3.2.2 Barrett’s modular multiplication algorithm

[Match Grade: A]

Algorithm:

**Input:** X (2n bits), P (n bits) [X is the already computed product.]

**Output:** A = X mod P

Step 0: Pre-compute \(\mu = [2^{2n} / P]\)

Step 1: Compute \(k = [X / 2^n]\)
Step 2: \( A = X - \lfloor k\mu/2^n \rfloor \times P \)

Step 3: While \( A > P-1 \),
\[ A = A - P \]

Step 4: Return \( A \)

Insight into the working:

The ranges of the operand to be reduced (X), and of the modulus (P) are well defined. This enables one to make certain pre-computations based on just the range of X, and the value of P. These types of pre-computations are especially useful when quite a few modular multiplications are to be performed with the same modulus.

The only pre-computation required for successful modular reduction using the Barrett’s algorithm is, \( \mu = \lfloor 2^{2n}/P \rfloor \). One might wonder about the choice of \( 2^{2n} \) in the numerator, but the reason that might best explain this choice would be truncation loss. To understand what is meant by truncation loss, let us look at an example.

Let \( n = 10 \). Then, \( 2^n = 2^{10} = 1024 \), and \( 2^{2n} = 2^{20} = 1,048,576 \). Suppose \( M = 591 \). (These are all randomly chosen numbers for the sake of illustration.)

\[
\begin{align*}
2^n/P &= 1.73 \quad \text{(A)} \\
\lfloor 2^n/P \rfloor &= 1 \quad \text{(A_T)} \\
2^{2n}/P &= 1774.24 \quad \text{(B)} \\
\lfloor 2^{2n}/P \rfloor &= 1774 \quad \text{(B_T)}
\end{align*}
\]

We can see that there is more information loss when (A) is truncated to get (A_T), than the loss when (B) is truncated to (B_T).
At the end of this fabricated example, it is natural to feel that it is beneficial to use bigger numerators (whose effect can always be negated by dividing later by a power of 2), to minimize the truncation loss. This could be done, but at the cost of having bigger multiplications while running the Barrett’s algorithm. So, one has to be familiar with the existence of this trade-off.

The same concept of optimizing truncation loss, while keeping the multiplier size to a minimum is a challenge not only in the pre-computation step, but also during the runtime of the algorithm.

So, it all boils down to managing two trade-offs, as we shall see in the following equation:

\[ q = \left\lfloor \left\lfloor \frac{X}{2^n} \right\rfloor \mu / 2^n \right\rfloor \]

where, \( q \) is the estimate of the quotient.

\[ A = X - qP \]

Then, \( P \) has to be subtracted from \( A \) till the result is smaller than \( P \). The number of such subtractions does not usually exceed two.

The resultant \( A \) is the reduction result.

It is not hard to get a feel for the working of the Barrett’s algorithm. The sole intention of this algorithm is to come up with an estimate of the quotient, without having to divide by the modulus \( P \). \( \left\lfloor \frac{X}{2^n} \right\rfloor \) gives a number that is lower than the correct quotient. That is because the divisor is \( 2^n \), which is greater than \( P \). But if this intermediate quotient is
multiplied by $2^n/P$, and then truncated, the resulting product should be a pretty accurate estimate of the quotient. This understanding comes from the *chain rule*. But as we saw above, the truncation loss is too much when we divide $2^n$ by $P$. Therefore, we divide $2^{2n}/P$, truncate it, and later divide the product of $\mu$ and $\lfloor \frac{X}{2^n} \rfloor$ by $2^n$. The final result is again truncated, since the estimate of the quotient has to be an integer.

This provides the intuitive explanation of the working of the Barrett’s algorithm.

Cost analysis:
The pre-computation cost (which applies to the computation of $\mu$) is usually not taken to be part of the cost. However, just for completeness, we can see that it takes a division of a $2n$-bit number by an $n$-bit divisor. If this pre-computation were part of the run-time computation, it could easily spell disaster for the Barrett’s algorithm.

Run-time cost: Barrett’s algorithm needs $2kxn$ multiplications (where $k$ is at most equal to $n$).

### 3.2.3 Modified (Folding) Barrett algorithm

[Match Grade: A]

Insight into the working:

The main idea behind the folding Barrett algorithm is to partially reduce the operand $X$ to $X’$. This initial reduction is done by the folding. Then, the classical Barrett algorithm is applied to $X’$, instead of $X$. Since $X’$ is smaller than $X$, this arrangement brings about reduction in computation.
The trade-off one should be wary of is the amount of folding versus run-time computational cost. There is a *point of diminishing returns* for the folding mechanism. What this means is that after this point, the computational effort that goes into folding is not justified by the reduction in the computational cost of the regular Barrett algorithm that follows folding.

Let us see how this point can be worked out.

Consider a system which has $F$ folds. For each fold, we need to compute $P^{(i)} = 2^{(1+2^{-i})n} \mod P$. So, for fold 1 we have $P^{(1)} = 2^{1.5n} \mod P$; for fold 2, $P^{(2)} = 2^{1.25n} \mod P$; for fold 3, $P^{(3)} = 2^{1.125n} \mod P$, and so on. This leaves the classical Barrett algorithm with a pre-computation, $\mu = \left\lfloor \frac{2^{(1+2^{-F})n}}{P} \right\rfloor$.

$$A = X^{(F)} - \left\lfloor \frac{X^{(F)}}{2^{(1+2^{-F})n}} \frac{\mu}{2^{(2-F)n}} \right\rfloor P$$

$X^{(F)}$ is arrived at in the following way:

```
LOOP (i = 1 to F)
    X^{(i)} = X^{(i-1)} \mod 2^{(1+2^{-i})n} + \left\lfloor X^{(i-1)} / 2^{(1+2^{-i})n} \right\rfloor P^{(i)}
```

where, $X^{(0)} = X$.

It is worked out in [29] that the optimum number of folds ($F$) to minimize the run-time of the modified Barrett algorithm is 1.44. Since $F$ has to be an integer, it can be taken that it is not beneficial to fold more than twice, and that at least one fold will yield better results than not folding at all.
Let us see why the technique of folding gives us the desired answer, with the aid of a representative example. Suppose, \( X = (177)_{10} \) and \( P = (10)_{10} \). The bit-length of \( P \), which is denoted by \( n \), is 4. As expected, the bit-length of the operand \( X \) is \( 2n = 8 \). Let \( A \) be the remainder, which is to be computed.

Let us see the effects of folding once, and then twice. The first fold occurs at \( 2^6 = 64 \).

Pre-computations: (1) \( P' = 2^6 \mod P = 64 \mod 10 = 4 \)

\[ (2) \mu = \lceil \frac{2^6}{P} \rceil = \lceil \frac{64}{10} \rceil = 6 \]

Partial reduction: \( X' = X \mod 2^6 + \lfloor X/2^6 \rfloor P' = 177 \mod 64 + \lfloor 177/64 \rfloor 4 = 49 + 8 = 57 \)

We can immediately see that 57 gives the same remainder when divided by \( P = 10 \) as does 177. So, 177 has been partially reduced to 57, and the classic Barrett algorithm just has to work on a number much smaller than the original operand. In this example, we can see an almost \( \frac{2}{3} \) reduction in the operand.

An intuitive way to understand how the folding works is as follows: Suppose \( B \) is fully divisible by the modulus \( P \). Now, assume we scoop out \( B \) from the operand \( X \), \( k \) times. In effect, what we have now is \( X' = X - kB \). \( X' \) has to give the same remainder as \( X \) would, when divided by \( P \) — since \( B \) is a multiple of \( P \). To make sure \( k \) takes the maximum possible value (to leave \( X' \) positive and less than \( B \)), we can redefine \( X' \) to be \( X \mod B \). But often, making use of just any \( B \) is not worth the extra step at all. However, it is a
totally different story when B is a power of 2, in which case X’ mod B becomes just a case of bit masking – which is almost free on hardware. But, choosing B as a power of 2 leads to another problem: this power of 2 may not be a multiple of P. (If P is a prime number – as the case frequently is – a power of 2 is never a multiple of P.)

To solve this problem, we need to first calculate P’ = B mod P. We need one more piece of information – we need to know how many times we have scooped out B from the original operand X. In other words, we need to know the value of \( \lfloor \frac{X}{B} \rfloor \). If we multiply this value with P’, we would have computed the offset. This offset would be zero if B were a multiple of P. Since B is chosen as a power of 2, we will have a positive value of this offset. If we add this offset value to X mod B, we would have successfully partially reduced the operand to an intermediate operand, X’, whose remainder characteristic with respect to the modulus P would be identical to the original operand, X.

Now, just to see what happens, let us try folding it once again. \( X^{(2)} = 57 \mod 32 + \lfloor \frac{57}{32} \rfloor \times 2 = 27 \). We can verify that, \( 177 \mod 10 = 57 \mod 10 = 27 \mod 10 = 7 \).

3.2.4 Montgomery and Barrett reduction-based interleaved modular multiplication

[Match Grade: A]

This work [31] is one of the latest advances in the area, which makes use of Mersenne primes as the basis for moduli selection. Their paper also recognizes that Montgomery and Barrett algorithms need single-precision multiplications, which can be conveniently
converted to shift operations – by keeping the pre-computed values Mersenne (unity-short of a power of two).

The authors propose four sets of moduli, out of which we limit our interest to two sets of moduli defined specifically to speed up Barrett’s algorithm.
4.1 Methodology

A broad-based approach to solving the modular multiplication problem defined in the previous section is to analyze all components of the problem separately, and then check if there is a fast algorithm to speed up each component. Achieving a fast modular multiplier may not be as simple as simply bringing together the sped-up components, which gives rise to concerns over algorithmic compatibility. Therefore, it is prudent to decide whether to adopt an integrated approach (that makes use of interleaved partial multiplication and partial reduction) or a serial approach (multiplication followed by reduction), before going into the specifics of the algorithms and the scope for improvements thereof.

A major reason to favor the serial approach is the freedom in choosing algorithms for each part, independent of the other. As long as it is made sure that the output of the multiplier section forms a compatible input to the reduction section, this independence is certainly a sought-after feature. However, another factor to be considered is the impact of this choice (of choosing the serial approach over the integrated) bears on the performance of both sections together. Cetin Kaya Koc, Tolga Acar, and Burton Kaliski Jr. have shown in [30] that the serial approach (which they refer to as the “Separated Operand Scanning” or simply SOS) is nearly as fast as the interleaved approach (which they refer
to as the “Coarsely Integrated Operand Scanning” or just CIOS). This makes it rational to go with the serial/separated approach. In this thesis, it is assumed that the multiplication has been performed in the fastest way possible, and we have the product ready to be reduced. This assumption makes the reduction completely parallel, rather than bit-serial or word-serial.

A. THE MODULI SETS

It is to be noted that the modular operations being performed are in prime Galois fields, GF(p). The modulus (represented by M henceforth) in consideration would then be a prime number. The problem statement puts an additional constraint on the modulus in terms of the maximum length, restricting the prime modulus to $n$ bits, i.e., $0 < M < 2^n$, such that $\prod_i d_i(M) = M$, where $d_i(X)$ represents the $i^{th}$ positive integral divisor of the argument $X$.

The above representation is the most basic, broadest set of possible moduli, which is essential to the problem statement. However, going beyond the essentiality of this definition, researchers have continually opened up paths to speed up the reduction in GF(p). The most recent and significant example of constricting the permitted moduli sets can be seen in [31], wherein four sets of moduli are defined on the basis of speeding up stemming out of Mersenne properties – two each for Barrett-based reduction and Montgomery-based reduction. We present in this paper four moduli sets, different – both
in representation and rationale – from those presented in [31]. Our proposed moduli sets and algorithm are based on strict and looser views of Mersenne numbers.

Let us start with the strict view of Mersenne numbers, which we shall refer to plainly as Mersenne. Consider a Mersenne number, \( M_r \). By definition, \( M_r = 2^r - 1 \), where \( r \in \mathbb{N} \). A widely known property of Mersenne numbers is that \( M_r \) is prime for a prime value of \( r \). For the sake of convenience, let us assign \( r \) an open value of \( p \), which just indicates that a prime value is assigned to \( r \). The notation we shall adopt in the rest of the paper is \( M_r \) for a general Mersenne number (which may or may not be prime), and \( M_p \) for a Mersenne prime.

At this stage, we will introduce the four sets of moduli we recommend for speeding up Barrett-based reduction, and follow up with a short discussion of the density function of each of these sets, in lieu of a lengthier treatment and illustration of the mathematics behind the choice of these sets, in view of brevity due to the space restrictions.

Set 1: \( M = 2^p - 1 = M_p^{[s]} \) \hspace{1cm} (1)
Set 2: \( M = d(M_c) = M_p^{[l]} \) \hspace{1cm} (2)
Set 3: \( M = 2^k - 2^l \pm 1 = Q_p^{[s]} \) \hspace{1cm} (3)
Set 4: \( M = d(Q_c) = Q_p^{[l]} \) \hspace{1cm} (4)

where, \\
\( M_p^{[s]} \) is a strict Mersenne prime.
$M_c$ is a Mersenne composite, i.e., $M_c \in \{M_r\} - \{M^{(s)}_p\}$.

$M^{(l)}_p$ is a loose Mersenne prime, which is any integral prime divisor of a Mersenne composite.

$Q^{(s)}_p$ is a strict Quasi-Mersenne prime.

$Q_c$ is a Quasi-Mersenne composite, i.e., $Q_c \in \{Q_r\} - \{Q^{(s)}_p\}$.

$Q^{(l)}_p$ is a loose Quasi-Mersenne prime, which is any integral prime divisor of a Quasi-Mersenne composite.

These four sets together constitute a fairly large number of prime numbers, thus ensuring there is sufficient choice in the design of the system. The following note on the density functions of each of the sets will serve as an intuitive aid in understanding the coverage of primes by these sets.

Let us start with a general assumption that the highest allowed value that a modulus can take is $V = 2^n - 1$, making $V$ the highest possible Mersenne number attainable by the modulus. The number of Mersenne numbers up to $V$ (including $V$) is $n$. Out of these $n$ Mersenne numbers, only $\pi(n)$ are prime [32]. Therefore we can report the cardinality of the strict Mersenne prime set as,

$$\#(M^{(s)}_p) = \pi(n) \quad (5)$$

Note that:

$$\pi(n) \cong Li(n) = \int_2^n \frac{dx}{\ln(x)} \quad (6)$$
Though (6) represents the generally adopted form of the logarithmic integral in prime counting, a faster convergence may be achieved [33] through:

\[
Li(n) = \gamma + \ln \ln(n) + \sqrt{n} \sum_{i=0}^{\infty} \frac{(-1)^{i-1} (\ln n)^i}{i! 2^{i-1}} \sum_{k=0}^{[(i-1)/2]} \frac{1}{2k+1}
\]  

(7)

where, \( \gamma \) is the Euler-Mascheroni constant.

\[
\gamma = \lim_{t \to \infty} \sum_{k=1}^{t} \frac{1}{k} - \ln t = 0.57721566 ...
\]  

(8)

Next, let us estimate the number of loose Mersenne primes less than \( V+1 \). First, let us note the number of Mersenne composites less than \( V+1 \),

\[
\#(M_c) = n - \pi(n)
\]  

(9)

In order to estimate the distinct, non-repeated prime divisors of the Mersenne composites, it would be useful to reduce the set of Mersenne composites to a set \( H \) of \( h (\leq \#(M_c)) \) co-prime numbers by iterative application of the parallelized I-G Binary GCD algorithm, which is up to eight times faster than the traditional Euclidean approach [34].

\[
\#(M_p^{(l)}) = \sum_{i=0}^{h-1} \omega(H_i) \mid \text{such that } d_j(H_i) \notin \{M_p^{(s)} \}
\]

\[
\prod_j d_j(H_i) = H_i, \forall \ i, j
\]  

(10)

where, \( \omega(X) \) gives the number of distinct prime divisors of the argument \( X \).
We now need to estimate the number of strict Quasi-Mersenne primes. This is trickier than the previous cases, mainly due to the unmanageable number of combinations and prime tests thereof. However, we can see that Proth primes (i.e., primes of the form: \(k_1 \cdot 2^{k_2} + 1\), where \(k_1 < 2^{k_2}\)) make up a considerable chunk of the \(Q_p^{(s)}\) set. This can be easily visualized by plugging a difference of two powers of 2 into \(k_1\). Fortunately, a Proth number can be rather easily checked if it is indeed a prime number, as shown in [35]. Then, we have the Solinas primes [36], of the form \(2^a \pm 2^b + 1\). If \(#(P_p)\) indicates the number of Proth primes below \(V+1\), and if \(#(S_p)\) indicates the number of Solinas primes below \(V+1\), we have the lower and upper bounds on the number of strict Quasi-Mersenne primes below \(V+1\).

\[
#(P_p) \leq #(Q_p^{(s)}) \leq #(S_p) \tag{11}
\]

[37] contains a section on counting the Solinas primes.

On similar lines of quantifying the number of loose Mersenne primes, we can reduce the Quasi-Mersenne composites from \(#(Q_c) = n^2 - n\), to a set \(G\) of \(g \leq #(Q_c)\) co-prime numbers (again via iterative use of the parallelized I-G Binary GCD algorithm). Then, we simply pick the distinct prime factors of each element of \(G\).

\[
#(Q_p^{(l)}) = \sum_{i=0}^{g-1} \omega(G_i) \mid d_j(G_i) \notin \{Q_p^{(s)}\}; d_j(G_i) \notin \{M_p^{(l)}\}; \quad d_j(G_i) \notin \{M_p^{(s)}\}; \quad \prod_j d_j(G_i) = G_0, \forall i, j \tag{12}
\]
Let \( \#(P_V) \) be the number of primes lesser than \( V+1 \), such that they belong to one of the four sets outlined in (1), (2), (3), and (4). Then, we have,

\[
\#(P_V) = \#(M_p^{[s]}) + \#(M_p^{[l]}) + \#(Q_p^{[s]}) + \#(Q_p^{[l]})
\]

(13)

The physical interpretation of (13) is simply that we have \( \#(P_V) \) number of choices to pick a prime number from as the modulus for the multiplication and the subsequent reduction. The forthcoming algorithms are framed with these four sets of moduli in mind, and are shown in the last section to be faster than the state-of-the-art algorithms.

B. THE ALGORITHM

Input: \( A, B, M \)

Output: \( R \)

Pre-computations:

1. \( N = A \cdot B \)
2. \( F = 2^k, k \in \mathbb{Z} \)
3. \( M_1 = F \mod M_{\text{sets} 1,2} = 1 \)
4. \( M_2 = F \mod M_{\text{sets} 3,4} = 2^l + 1 \)

For Moduli Sets 1 and 2:

Step 1: \( R_1 = N[k - 1: 0] \)  \hspace{1cm} // N mod F
Step 2: $R_2 = N[2n - 1:k] \quad // [N/F].M_1$

Step 3: $R = R_1 + R_2$

Step 4: Return $R$.

For Moduli Sets 3 and 4:

Step 1: $R_1 = N[k - 1:0] \quad // N \mod F$

Step 2: $R_2 = (N[2n - 1:k]|L) \div N[2n - 1:k]$

$\quad // [N/F].M_2$

Step 3: $R = R_1 + R_2$

Step 4: Return $R$.

Defining variables used in the algorithm:

$A$: The multiplicand, $2^{n-1} \leq A < 2^n$.

$B$: The multiplier, $2^{n-1} \leq B < 2^n$.

$N$: The product to be reduced, $2^{2n-1} \leq N < 2^{2n}$.

$M$: The modulus, $N < 2^n$.

$R$: The partially reduced result.

$k$: The smallest integer such that $x$ is minimum.

$\alpha[i:h]$: The portion of any variable $\alpha$, between and including its $i^{th}$ and $h^{th}$ bits.

$L$: An $l$-bit zero vector

Brief analysis of the algorithm:
The multiplication is put under pre-computation since the focus of the algorithm is the modular reduction which follows the multiplication. Step 1 computes the remainder of the division of the product and the power of 2 (F), which is just a case of masking the higher order bits, while choosing only the lower k bits. This gives us the value of N mod F. Step 2 could either fold once more, or make the adjustment to reduce the result with the actual modulus. In case of a second fold (which is applicable only to moduli of sets 3 and 4), the unity adjustment occurs later in the step. In effect, Step 2 would give us the value of $\lfloor N/F \rfloor M_{1 or 2}$. Step 3 is basically the summation of the results of the first two steps, which gives the partial reduction. It is this result which is fed into the classical Barrett algorithm.

As we can see, in case of Moduli Sets 3 and 4, one addition operation and a shift (left shift by $l$ bits) are introduced. In comparison with the multiplication operation existing in current algorithms, even Moduli Sets 3 and 4 are much inexpensive. Moduli Sets 1 and 2, on the other hand, are next to free – when it comes to hardware costs.

The cost involved during the run-time of the algorithm is a maximum of three shift operations, bit-masking, and two $s$-bit addition operations (where $s$ is at most 20% longer than the modulus length).
4.2 Results

The performance of the proposed algorithm is compared with that of the leading algorithms that focus on modular reduction for large-integer multiplication. It needs to be clarified upfront that performance can have several aspects to it, like speed (or equivalently time taken), silicon area, and power consumption, for instance. Since this is a cryptographic setup, not much attention (beyond maintaining state-of-the-art figures) is given to area and power aspects, as much as for the speed.

The platform used for comparison is Altera Stratix III FPGA, and the design has been carried out on Altera Quartus II. Comparing across platforms could be unfair to the proposed algorithm or to the ones compared or to both. Hence, the algorithms being compared have been accurately and carefully implemented on Altera Stratix III, alongside my own implementation [38, 42].

The testing has been carried out in a systematic manner, with 10 unbiased representative operand-modulus pairs for each bit-length reported in the test. The reported bit-lengths are 100, 200, 300, 400, 500, 600, and 700, thus making use of 70 such distinct operand-modulus pairs. The values reported in this thesis represent the average time measurements per bit-length category.

Before presenting the results, let us qualitatively see why the algorithm has the potential to produce faster modular reduction. It is clear that reducing any large number with a
power of 2 as the intermediate modulus is a simple case of bit masking, and is a negligible hardware effort. Reducing the partially reduced result further with the actual modulus costs a single-precision multiplication. However, if the actual modulus is smaller than the intermediate modulus by unity (as defined by moduli set 1), the need for the multiplication vanishes, and is replaced by shift and addition/subtraction operations.

It should be noted that the actual modulus need not just be unity short of the intermediate modulus; it may alternatively be an integral prime divisor of unity less than the intermediate modulus, thus leading to the moduli set 2.

The rule followed in testing the algorithm is that the length of the modulus is less than or equal to half the product length in bits.

For the purpose of comparison, we have chosen the Modified Barrett algorithm of [29], since it is faster than Montgomery and classical Barrett algorithms. The work of [31] has been compared at 3 bit-lengths: 192 bits, 256 bits, and 512 bits. This finds convenience in 32-bit word block usages.
Fig. 4: Time delay comparison

Fig. 5: Time delay comparison – smoothed graph.
The main sources responsible for the heavily enhanced speed-up are (i) lack of multiplication operations in the proposed algorithm, and (ii) parallel-input scheme employed for treating the pre-computed product.

It has been made evident in this thesis that by defining moduli sets based on Mersenne, Quasi-Mersenne, and divisor primes thereof, and by updating and tuning the Modified-Barrett algorithm presented in [29], we achieve much better speed and significantly improved levels of reduction.

Fig. 6: Relative speed-up (as a percentage).
CHAPTER V

CONCLUSION AND FUTURE WORK

It is time to pull the classic fluorescent highlighter out, and mark up my thesis into a concise statement, starting with the problem chosen, the deficiencies of the existing solutions, my proposed solution, why my proposed algorithm works, and finally scoping the possibilities for future work.

5.1 A summarized view of the thesis

After an extensive technical overview of large-integer modular multipliers, I narrowed down the options for potential contributions to 3 main areas:

(i) Classic Multipliers (non-finite-field that is)
(ii) Multiply-and-reduce alternators (word-serial interleaved algorithms)
(iii) Modular reduction units (assuming the multiplication has been performed in the most efficient way possible)

Route (i) was not chosen due to the heavy amount of work that has already been carried out in this topic, rendering it a nearly-saturated problem to work on. Route (ii) was not chosen as a way to keep maximum flexibility in the multiplication and reduction algorithms, so as to keep them as less intertwined as possible. Apart from that reason too, the word-serial nature of most solutions in this route tend to slow down the computation – in comparison with parallel processing. Choosing a fast multiplier and working on route
(iii) seemed to be the best option, in terms of scope for contribution, within the framework and time constrictions of a typical master’s degree.

The main existing solutions are the Montgomery algorithm, the Barrett algorithm, and the Modified Barrett algorithm proposed by Hasenplaugh, et al. The Montgomery algorithm was a ground-breaking work, tracing back to 1986. The algorithm speed is nearly constant, which is a huge disadvantage for operands smaller than 1024 bits. Even for operands bigger than 1024 bits, the Modified Barrett algorithm outperforms the Montgomery results.

Barrett algorithm is faster than Montgomery algorithm initially, but does not pay off as the operand length mounts. However, the Modified Barrett algorithm was developed specifically to plug the very issue. The most visible benefit of Barrett algorithm though is the absence of any requirement to compute modulo-residues, which happens to be plague the Montgomery algorithm in a serious way – especially for fewer operations per modulus.

The Modified Barrett algorithm outperforms both the Montgomery and Barrett algorithms. However, it came to our attention that the formulation of the Modified Barrett solution is not completely optimized. To put it simply, the main cost of the Modified Barrett algorithm can be attributed to a single-precision multiplication.
The proposed algorithm reduces this multiplication to at most an addition operation and a shift operation, and in the best case, to just a shift operation. All the speed-up reported in this thesis is essentially a result of this elimination of the multiplication operation. This is precisely the reason why my proposed algorithm works faster than all of the previously published work.

My main contribution lies in defining 4 moduli sets (based on Mersenne and Quasi-Mersenne properties) which are crucial in reducing the multiplication operation to an addition operation and/or shift operations. This contribution is made at a fundamental level, and has amplifying impact on speed all the way up to modular exponentiation, which lies at the heart of RSA performance.

On an average, the proposed algorithm has an asymptotic speed betterment of more than 80% compared to the Modified Barrett algorithm. The asymptotic behaviour is approached as early as 600 bits, and the least speed-up itself is about 70% faster than the run-time of the Modified Barrett algorithm. All testing has been carried out on Altera Stratix III FPGA, and implementation designs made use of Altera Quartus II. Area and power dissipation are not increased, thus avoiding any expected trade-off.
5.2 Scope for future work

Though much thought and effort have gone into this master’s work, lack of additional time leaves scope for extending this work into very useful and interesting territory.

(i) The moduli sets can be further extended. Though this effort may not lead to further speed-up, it surely gives more option to the RSA system designer in terms of moduli selection.

(ii) In this present work, reducing the area and power dissipation was not a concern. However, I have just made sure that those two critical parameters are not increased. Therefore, as an augmentation to my work, area and power dissipation could be reduced without affecting the speed.

Area and time are usually in a competing mode, which could be confusing at first – when the thesis recommends reducing both time and area. So what is really meant here is that a commendable work in the future could try to eliminate one of the blocks of operation – which automatically cuts area requirements and the time consumed by that block.

(iii) My proposed modular reduction unit can be applied to the multiply-and-reduce route, since word-serial approaches have their own advantages (especially in reducing silicon area).

(iv) This work can be featured on hand-held computation devices, with just some attention to the power consumption. The motivation for this suggestion comes from observing the new trend of super-powerful processors on tiny devices (up to 1.5 GHz processors on the fresh crop of smart-phones).

(v) Biometric cryptography can be pushed closer to real-time performance with this newly gained speed-up.
REFERENCES

[1] P. J. Cameron, “Notes on Algebraic Structures”, *Course notes at the Queen Mary University of London*.


Match Grade: B

http://ozark.hendrix.edu/~burch/proj/karat/results.html
Match Grade: B

Match Grade: C

Match Grade: C
Match Grade: B-

Match Grade: B-

Match Grade: B-

Match Grade: C

Match Grade: C

Match Grade: B


Match Grade: B+


Match Grade: B


Match Grade: B+


Match Grade: B+

Match Grade: B+


Match Grade: B


Match Grade: A


Match Grade: B+


Match Grade: B


[26] H. Wu, “Bit-Parallel Finite Field Multiplier and Squarer Using Polynomial


VITA AUCTORIS

Suhas Sreehari was born in Bangalore, India, and the landmark event is consistently traced back to a shady Sunday of September, 3 years shy of 1990.

After the typical schooling drill, he went on to perform many mischievous and notorious activities. Listing all of them would not bring any noticeable glory to Suhas or to this thesis; the benefit brought in terms of the additional pages the Vita Auctoris would contribute to this thesis effortlessly is far outweighed by the boredom in bragging about oneself – and after heavy consultation was deemed too risky to further irritate the graduate committee members after a pretty long-drawn pointless 73 pages of mere formality, whose succinct presentation would not take more than 4 pages; 5 tops with the torturous tradition of continuously glorifying other researchers – either dead, or foggily old, or freshly beaten by Suhas’ present work, telling them “You did this, and I know. Yes, I know it was you. Ah, those were your years!”, and many more such back-pats.

Anyway, that would certainly make case for a short Vita Auctoris.

Suhas had no intentions of graduating within the top 10% of his undergraduate class at the BMS College of Engineering (VTU, India), nor was securing distinction planned. What started as an innocuous glorification of previous modular reduction algorithms somehow turned into a much better algorithm, and Suhas is deeply apologetic towards the aged researchers whose algorithms have just been outperformed. With this mishap, Suhas could not help but earn his master’s degree from the University of Windsor, Canada, with a cumulative GPA of 12.5/13.

The academic community is scared of the many more potentially unplanned things Suhas might go on to achieve. The last time we heard from him, he was storming into biomedical engineering, so stay tuned.