An Efficient And High Performance Architecture Design And Implementation Approach Of Cryptographic Algorithm Computation For Authentication In Modern Wireless Communication Applications

Lega.Karunankitha [a], Pericherla Sravya[b], G.Aparna[c], Surisetty Navya[d], P.Abhigna[e], Dr.B.Suneela[f]

[a] Assistant Professor, ECE Department, T.K.R Engineering College, Nagole, Hyderabad, Telangana, India
[b] Asst. Professor, ECE Department, Malla Reddy Engineering College (Autonomous), Hyderabad, Telangana, India,
[c].Research Scholar, ECE Department, University College of Engineering, Osmania University, Hyderabad, Telangana, India,
[d] Asst. Professor, ECE Department, Malla Reddy Engineering College (Autonomous), Hyderabad, Telangana, India,
[e].Assistant Professor, ECE Department, DNR College of Engineering and Technology, Bhimavaram, Andhra Pradesh, India,
[f]. Associate professor, Lords Institute of Engineering Technology, Hyderabad, Telangana, India

Article History: Received: 11 January 2021; Revised: 12 February 2021; Accepted: 27 March 2021; Published online: 10 May 2021

Abstract: According to the general reflections in the former section, the following goals are for this proposed work. The aim of the proposed work is implementation of the hardware architecture for L.M. Montgomery Modular Multiplication Algorithmic and proposes a mountable systolic architecture which is operative for large bit-lengths and it is appropriate for equally regularly used categories of open Key Cryptography (PKC) i.e. ECC and RSA Crypt-systems. The open key algorithms such as RSA ,Diffed-Hellman key exchange and Elliptical curve cryptography make use of high end computational operations like exponentiation which can be computed by using fast implementation of Montgomery modular exponentiation method. Here developing the Montgomery modular multiplication technique is aimed. The main aim of this procedure is to reduce the requirement of area and time because the less area x time factor is very useful for realizing in to hardware for real time applications.

Keywords: Real time Applications, Diffe-Hellman_Key, and Elliptical curve cryptography

1. Introduction

Modular multiplication is considered as the fundamental computational requirement in most of the open-key cryptograms like RSA, Diff-Hellman key agreements (DH), Elliptic Curve Cryptography (ECC). The proficiency of this modular multiplication has the ability to enhance the improvement of the cryptosystem. The trial division operation in the modular multiplication is the prime factor which plays a vital role in the performance measure. This has become a challenge to find a performing to overcome the trail separation operation. Among various algorithmic rules designed to perform this operation Montgomery Multiplication algorithm is the most popular as this regularizes the outcome of the multiplication and decreases the complication of the standard multiplication especially if biggest numbers are engaged like in open key cryptography (RSA). This algorithm is the proven to be quickest algorithmic rule to perform X*Y mod M for large values of X, Y and M respectively. For performing the modular multiplication two algorithms and their architectures were proposed they are optimizations of the L.M.Montgomery standard multiplication and the interleaved standard multiplication algorithmic rule. In the proposed work L.M.Montgomery standard multiplication architectures for computing the modular multiplication is implemented and the simulation and synthesis are done on Xilinx FPGA. The main advantage of the implementation is that it is extensible to at all accuracy of that given input numerals of X,Y&M. the optimization of the multipliers is evaluated and compared founded on the architectures in terms of area-time product factors. It is observed that the enhanced algorithmic and their designing need very less firmware and attains high motion of calculation when comparable with multipliers with the earlier L.M.Montgomery algorithmic rule.

1.2 INSPIRATION

It is extensively accepted that safety matters will show a critical role in various future computer and communicating system's. A vital instrument for achieving systems safety is cryptography. For improvidence as well as for bodily safety motives it is often needed to actualize cryptography algorithmic rules in firmware. Conventional ASIC solutions, thoroughly have the renowned shortcoming of deduced tractability comparing to freeware problems. Since standard safety protocols are progressively outlined to be algorithmic in dependent, a elevated degree of tractability with regard to the crypto graphic algorithmic is looked-for. A promising result which combining huge flexibleness with the fastness and personal safety of conventional firmware is the application of cryptography algorithmic on reconfigured devices such as FPGA's & EPLD's. In the case of open-key scenario's, algorithmic in dependence can mean not only a modification of the exact crypto graphic algorithms but also a modification of variables such as bit length, modulus, or exponents. One solicitation, deals with in this statement, including arithmetic architectures for standard augmentation with very longer integers which are at the heart of up-to-the-minute public-key schemes. Most markedly, RSA and distinct logarithm-based (that is., Diffied-Hellman key exchange or the Digital Signature Algorithm, DSA) schemes need standard durable variable exponentiation.

The escalating evolution of information communiqué and electronic proceedings over the internet has made safety to turn out to be the chief concern through the network. To offer current security features, open-key crypto-systems are used. The broadly usable algorithmics for open-key crypto-systems are RSA, Diffien-Hellman key agreement of DH, the numerical Signature Algorithm (DSA) and systems constructed on elliptic curve crypto graphy (ECC). Altogether these algorithmic rules have main article in commonly: that is those function on very vast numbers (that is 160 to 1024 bit)

Lengthy term duration are essential to deliver an adequate extent of safety, but also relation for the procedure value of these algorithmic rules. So far, the maximum prevalent open-key scenario in use currently is RSA. The central maneuver for information encoding processing in RSA is standard involution, which is done by a sequence of standard propagations (that is X*Y modM). These accounts for most of the complexness in terms of period and resources required. Regrettably, the big word duration (example. 512 / 1024 bits) so the RSA system lag and problematic to utilize. This provides motive to find for devoted firmware results which calculate the standard multiplications proficiently with least resources. The L.M.Montgometry multiplication algorithmic rule is measured to be the profligate algorithmic to calculate X*Y mod M in processers having the standards of X,Y&M are big. An other cost-effective algorithmic for standard multiplication is the interleaved standard multiplication algorithm.

The task at hand is to plan arithmetical designing for operands with up to1024 bits on existing FPGA's. The precise lengthy word durations disallow the utilization of many suggested architectures as they should lead to unrealistically huge assets necessities. The standard exponentiation designing which mix L.M.Montgometry's standard deduction scheme and novel systolic array designs. The systolic array design involves well lesser logic references than several another systolic array designs for standard calculation. This is vital, as one of our objectives was to obtain results that can fit into a on its own FPGA, a architect goal that has lots of cost and designing benefits over several–FPGA outcomes. An other significant subjective was to systematically apply different architecture options for various bit duration and comparing the presentation and resource procedure.

1.3 ORGANIZATION OF PROPOSED WORK

2. Basic Terminology In Literature

2.1 MODULARMULTIPLICATION

Standard Reproduction is the numerical procedure on variables X,Y mod M with X,Y < M, where X , Y are the variables where M is modulus. In existing real-world crypto utilizations X,Y &M are big numerals of 150th bit or even-higher. They are several algorithmic for standard multiply operation. In this study, we emphasize on the 2 chief ones. Throughout intersperse standard propagation, the multiply operation and the computation of the remainder of the partition are interwoven. The duration of the intermediary outcome is of 1 / 2 bits wider compare to the operands is the benefit. The drawback is the usage of deletions in order to decrease the in-between outcomes. More meticulous data about the in-between standard propagation is in Part 2.1.1. L.M.Montgometry's standard multiply operation is the vast recurrently applied algorithmic for standard operation. The calculation is ended in the L.M.Montgometry domain. The benefit of this computation is that there is no necessity of reductions in order to condense the in-between outcomes. The shortcoming is the point that the L.M.Montgometry's standard multiply operation computes $X.Y.2^{-n}$modM instead of X.Y mod M and two L.M.Montgomery's standard multiply operations are needed for 1 standard propagation. Moreover thorough collection data about L.M.Montgometry's modular multiply operation is given in Section Part 2.2.2.

2.1.1 Interleaved Standard Multiplication

The concept of interleaved standard multiply operation is very easy: the 1st one is multiplied with the 2nd one bitwise and addend to the in-between outcome. The in-between outcome is condensed accordingly to the modulus. So this, 2 reductions per repetition are needed. Algorithmic 1 stretches a pseudo code operation of providing standard multiply operation.

Algorithmic 1 Interspersed standard Multiplication

```
I/P:X,Y,M with $0 \leq X,Y \leq M$
O/P:P = X .Y mod M
n:number of bits of X
xi:ith bit of X
1). P = 0;
2). for (i = n - 1; i ≥0; i = i -1)
3). {
4).  P = 2.P;
5).   I = xi .Y ;
6).  P = P + I;
```

It has different dis-advantages: The foremost concern is the comparing of P , M in the stages (7) and (8). In the bad instance every bit of M & P should be comparable. This can be resolved by an estimated evaluation with 2n rather than M. The next trouble is the quantity of add-ons or deductions in the steps (6), (7), and (8). All these computations can be substituted by single add-ons only. To do that, we pre estimate the amount of M's to be compute, and to calculate if Y would be add-ons in the coming process of the looping. There are less probable denominations for this valuation. These can be pre computed and kept in a look-up table. In each repetition of the circle, the approximation of the former loop can be added to the in-between outcome. Also, we can substitute the laggard non- superfluous add-ons by the quicker superfluous carry-save-addition. This instance complexness of a carry save add-ons is equivalent of the period lag of 1 single full add-on. The data-flow picture in Fig1 shows this improvement. The excess in-betwwen standard propagation needs one carry-save-addition only. Its whole time difficulty is equivalent to n time lagging of single full- add-ons.

2.1.2 Montgomery Modular Multiplication

Algorithmic rule 2 give a pseudo code execution of L.M.Montgometry's standard multipliy operation. A standard multiplication for numbers in standard depiction involves 2 modular-Montgomery-multiplications.

Algorithmic 2 Montgomery Standard Multiply operation.

```
    I/P: $X;Y < M < 2^n$, with$2^{n-1} < M < 2^n$ and M=2t
 + 1,
    O/P: P=X _ Y _ $2^{-n}$ modM.
    n: number of bit in X,
    xi: $i^{th}$ bit in X
    1).  P=0;
    2).  for (i = 0; i < n; i + +)
    3) {
    4).  P=P + xi .Y
    5).   if ($P_{0=}$ 1) P = P +M;
    6).  P=P div 2;
    7)  }
```

The benefit of Algorithmic is that it does not need any subtractions. But, the algorithm needs 2 add-ons per repetition of the loop, and these add-ons are lazy, as they are non-redundant. The standard additions were substituted by carry-save -additions. This type involves two carry save add-ons per repetition of the loop itself. The time complexness of each repetition is equivalent to the time break of 2full adders. As a result, the time difficulty of one Montgomery Standard multiply operation is equivalent to 2n time lagging of single full add-ons. One of 4 variable standards (0,M,Y, and Y+M) can be addend for every repetition of the loop. All these values can be stored a look-up table. The analogous value can be chosen and added to the halfway outcome. The benefit of this algorithm is its short instance hardness of n period lagging of single full add-on for the Montgometry standard multiplies operation.

2.2 L.M.MONTGOMERY MOULAR MULTIPLICATION

In 1985's Peter Montgomery presented a new technique for standard multiply operation. The method of L.M.Montgomery escapes the time overwhelming trial division that is a frequent holdup of other algorithmic. His technique is stated to be very effective and is the root of more applications of standard multiplies operation, in both freeware and firmware. In this proposed work we are looking for an expeditious hardware implementation of the L.M.Montgometry's standard multiply operation. An Effective execution of the MMM in firmware was considered by more authors. Systolic array design is one of the possibility for applications of open key cryptography in firmware. Different methods for systolic arrays were suggested, our aid is in merging a systolic array construction, which is expected to be the best select for firmware on existing (IC's), with the MMM in

Field-Programmable-Gate Array (FPGA). In this section we says that the execution outcomes of a standard exponentiation which we intended using our MMM circuit. The operation values for RSA using MMM can be determined. But, their answer is very inefficient as they used additional step in the main algorithmic for the MMM; this protocol was necessary since they did't use the ideal bound for the key limitation R (so-called Montgometry limitation). The standard involution algorithmic is normally repeating standard propagation of 1500 periods for 1024-bit operands.

2.2.1 MONTGOMERY REPRESENTATION AND MULTIPLICATION

The Montgomery representation defined as: the given three integers are x, m, R. If $0 \leq x < m < R$ and gcd (m, R) =1 then the representation is Rep(x)=x.R mod m. (since gcd (m, R) =1, Rhas inverse R^{-1})Then Rep(x) has inverse (mod m) as $Rep^{-1}(x_R) = x_R.R^{-1}$ modulus m=x mod m. And thus it can both enter and exit the representation. But note that neither Rep nor Rep^{-1} should be calculated the way the equations suggest. If $0 \leq x_R$, $y_R < m$, the Montgomery multiplication of x_R (×) y_R.

Montgomery Representation and Basic Algorithm:

The following data represents the Montgomery standard propagation of 2 operands:

C=A.B modM

Integer domain: Montgomery domain:

A $A' = A.\ 2^k$ modM

B $B' = B.\ 2^k$ modM

C'=MP(A',B',M)

$= A'.B'2^{-k}$ modM

$= (A.\ 2^k)\ (B.\ 2^k)\ 2^{-k}$ modM

$= A.B.\ 2^k$ modM

C $C' = C.\ 2^k$ modM

In the above representation A, B, C indicates three operands, M is modulus and k is number of bits (k=0, 1, 2…k-1).Then Compute MP (A B) $=A.B.2^k$ modn,such that A, B are k-bit numbers and

$2^{k-1} < n < 2^k$.Aditionally, gcd $(n,2^k)$ =1.

- Let A= $(A_{k-1},A_{K-2},\ldots A_0)$
- A.B $=(A_0.2^0+A_1.2^1+\ldots\ldots\ldots+A_{k-1}.2^{k-1}).B$
- $2^{-k}A.B=2^{-k}(A_0.2^0+A_1.2^1+\ldots\ldots+A_{k-1}.2^{k-1}).B$
- $(A_0 2^{-k}+A_1 2^{-(k-1)}+\ldots\ldots\ldots+A_{k-2}\ 2^{-2}+A_{k-1}\ 2^{-1}).B$

  (Or)

  $(A_{k-1}\,2^{-1} + A_{K-2}\,2^{-2}+\ldots\ldots\ldots + A_1 2^{-(k-1)} +A_0\,2^{-k}).B$

This expression will be computed by the following algorithmic: This algorithmic contains two stairs;

Step-1 computes the multiplication of two operands A, B and A.B product multiplied by the2^{-k}, after that step-2 computes the mod M operation. The total algorithm result is A.B 2^{-k} mod M.

The algorithm:

```
Step-1:  t=0;for i=0 to k-1
             t=t+Ai.B;
             t=t/2;
This shall compute 2^-k A.B
The algorithm for "mod M":
 Step-2: t=0;
for i=0 to k-1
             t=t+Ai.B;
             t=t/2;
if (t>=M) => t=t-M
```

It is a technique for modular reducing a number in Z_mR to a number in Z_m under certain conditions. The essence is that we can reduce a product of two numbers in Montgomery representation (i.e., in Z_m). Given l.M.Montgomery reduction) M_R such that gcd (m,R)=1 and m<R. Then for any integer T: $0 \leq T < m$ R, the Montgomery deduction of T is given as M_R (T)=T R^{-1} mod m.The essence of Montgomery reduction is that we can reduce a product of two numbers in Montgomery representation. Let x, y ∈ Z_m and let x_R=Rep(x)=x.R mod m,y_R=Rep(y)=y.R mod m.

Then x_R , $y_R \in Z_m^2$ =>$M_R(x_{R.}\ y_R)$

=>M_R (Rep(x).Rep(y))

=>M_R ((x. R mod m) (y. R mod m))

=>(x. R mod m)(y. R mod m) R^{-1}mod m

=>(x. y) mod m .R

=>(x. y) R.mod m => Rep(x .y)

2.2.3 MODULAR EXPONENTIATION

A frequently used computation in cryptography is modular exponentiation. If, forinstance, we want to perform a^b(mod n).For a $\in Z_n^*$and b ∈N, then the easiest algorithmic rule is to repetitively multiply amodulo n b times. If b = 23, then the following sequence of equations yield thecorrect result with 22 modular multiplications:

Rn (a^2) = Rn (a · a)

Rn (a^3) = Rn (a · Rn(a^2))

Rn (a^4) = Rn (a · Rn(a^3))

.

.

Rn(a^{23}) = Rn (a · Rn (a^{22}))

This can be simplified considerably, and the followingsequence of equationsyields the correct result but requires only 7 modular multiplications:

Rn(a^2) = Rn(a · a)

Rn (a^4) = Rn (Rn (a^2) · Rn (a^2))

Rn (a^5) =Rn (a · Rn (a^4))

Rn(a^{10}) =Rn (Rn (a^5) · Rn (a^5))

Rn (a^{11}) = Rn (a · Rn (a^{10}))

Rn (a^{22}) = Rn (Rn (a^{11}) · Rn (a^{11}))

Rn (a^{23}) = Rn (a · Rn (a^{22})).

This method can be generalized and the resulting square-and-multiply algorithmas captured in 4.4.1 algorithm, it works for modular exponentiation in any (multiplicative)group. Let 'G'be such a set, 'a' an element in the set, and 'b'positive integer (i.e., b ∈N). If we want to compute a^bin G, then we must havea binary representation of the exponent b (i.e., b = $b_{k-1} \ldots b_1\ b_0$) and process this string bitwise from one end to the other. More specifically, we process the exponent from the most significant bit (i.e., b_{k-1}) to the lowest significant- bit (i.e., b_0). The another direction is also possible and works similarly. In Algorithm_ A, the variable's' is used to accumulate the result. The changeable is initially set to 1.The exponent isthen processed from(b_{k-1} to b_0). For each exponent bit, the value of s is squared and multiplied with 'a'if the bit is equal to one. Finally, the algorithm return's', and this value represents a^bin G.

Algorithm A The square-and-multiply algorithm:

```
(a∈G, b = $b_{k-1} \ldots b_1 b_0$ ∈N)
s← 1; for i = k − 1 down to 0 do
s← s · s
Ifbi = 1 then s ← s · a
```

Return s; (a^b)

Let's have a look at an example. Regarding the group $Z_{11}$and want to compute 7^{22}(mod11), then we need to initially write the exponent in binary representation (is., b = $(22)_{10}$= $(10110)_2$) & set's' to 1. The computation according to Algorithm 4. 3.1 are as follows:

$7^{(1)_2} = 1^2 \cdot 7 \equiv 7$ (mod 11)

$7(^{(10)_2} = 7^2 \equiv 5$ (mod 11)

$7^{(101)_2} = 5^2 \cdot 7 \equiv 3 \cdot 7 \equiv 10$ (mod 11)

$7^{(1011)_2} = (10)^2 \cdot 7 \equiv 7$ (mod 11)

$7^{(10110)_2} = 7^2 \equiv 5$ (mod 11)

In the first iteration, s is squared and multiplied with 7 modulo 11. The result is 7. In the second iteration, this value is squared modulo 11. The result is 5. In the third iteration, this value is squared and multiplied with 7 modulo 11. The result is 10 (or−1). In the fourth iteration, this value is square and multiplied with 7 modulo11. The result is again 7. Finally, in the fifth and last iteration, this value is squared modulo 11. The result is 5(consequently ,7^{22}(mod 11) equals 5).

2.3 INTRODUCTION TO RSA

Ron Rivest invented the 1st open key plan was established in 1977, Adi Shamir and Len Adleman at MIT. Now Rivestt-Shamir-Adleman (RSA) is the greatest widely used and applied open key crypto-system. The open key plan is built on using several keys, one for encoing and a another but connected for decoding. The overall design involves performing the remainder afterwards exponential and standard operation of bigger numerical.

The RSA is a widely deployed algorithm for open-key cryptography. RSA is frequently used in electronic commerce protocols. However, with advances of computing power worldwide, the required length of the keys is growing rapidly, and hardware acceleration becomes essential unless the system contains a high-end CPU. Execution of the RSA algorithm in hardware has some essential advantages over freeware-only solutions. For lower CPU's, hardware is the only solution with acceptable user experience, and on battery-powered devices, a hardware implementation extends the battery life, allowing a typical signature verification operation to be completed in less than a second. Due to its extremely small size and matching low power ingestion, the hardware RSA implementation modify standard open key cryptography on smart cards and RFID devices. The RSA includes Encoding and decoding has followed, for about plaintext block M and code text C: C = M^e modn; M = C^d mod n. Usually, it contains a 3rd -party to create a pair of open key and to dispense keys to sender and receiving. Both should recognize the value of n. The sender has the information of open key e, and only the receiving system recognizes the safe key d. So, a open key of (e, n) and safe key (d, n) produced by 3rd -party is discrete into sender and receiver individually. For this algorithmic rule to be acceptable for open-key encryped, these below necessities must be meet:

1. It is attainable to effort values of e, d, n that M^{ed} mod n = M1 for all M < n.
2. It is comparatively easy to perform M^e and C^d for all numbers of M < n.
3. It is impracticable to find d given e & n.

ver an insecure communication line. We have different ways in which Bob and Alice can achieve this task, base on the trouble of resulting the discrete algorithm trouble. In this segment we explain the RSA open key crypto system, the first designed and surely the best known of such systems. RSA is entitled preceding its (public) inventors Ron Rivest, Adi Shamir, & Leonard Adleman.

Bob	Alice
Key Creation	
Choose secret primes p and q. Choose encryption exponent e with $\gcd(e,(p-1)(q-1))=1$. Publish $N=pq$ and e.	
Encryption	
	Choose plaintext m. Use Bob's public key (N,e) to compute $c \equiv m^e \pmod{N}$. Send ciphertext c to Bob.
Decryption	
Compute d satisfying $ed \equiv 1 \pmod{(p-1)(q-1)}$. Compute $m' \equiv c^d \pmod{N}$. Then m' equals the plaintext m.	

Table 2.1: RSA key generation, encoding and decoding

The standard exponentiation is a shared action for scuffling and is used by numerous open-key crypto systems, such as Deffied and Hellman and the Rivest, Shamir & Adleman encodings plans, as encoding/decoding technique. RSA crypto-system involves of a set of 3 parts: a mod of almost 1024 bits and 2integers *d and e* called secure and open keys that justify that holding $T^{ed} \equiv T$ mod M. Obvious text T following $0 \leq T < M$. Messages here encoded using the open key as $C=T^{e}$mod M and distinctively decoded as T= CD mod M. So the similar action is use to execute in together procedures: encoding and decoding.

The modulus M is taken to be the production of 2 long prime values, say P,Q. The open key e is usually less and has lesser bits set (is bits = 1), hence the encoding part is comparatively profligate. The secure key d has as more bits as the modulus M and is selected so that de=1mod(p−1)(q−1). This system is protected as it is computing hard to determine p&q. It has been verified that it is difficult to break an RSA crypto system with a modulus of 1024-bit or even-more. The standard expanding implements standard multiplies operation recurrently. So the performance of open key crypto-systems is mainly established by the execution performance of the standard propagation and involution. As the variables are generally longer (i.e. 1024 bits or more), and so as to met instance needs of the encoding/decryption per formations, it is vital to endeavor to abate the number of standard multiply operations executed and to reduce the instant needed by one standard multiplication. Standard multiplication A×B modM can be computed in 2 many ways: multiplying, that is calculating= A×B; then deducing, is. R = T modM or providing the multiply operation and the reducing steps.

2.5 HOW RSAWORKS?

It will start by prime's, P&Q. Next select an encoding exponent comparatively prime to (P-1)*(Q-1). Then generating a decoding exponentd so that e*d==1 (modulu(p-1)*(q-1)). Thus compute =p*q. To encode a message, change your msg into a higher integer m. Thus compute $c=m^{e}$ (modulo n). Decoding the message is easy: compute c^{d} (modulo n), though a quicker reach utilizes known values of p&q with this Chinese Remainder Theorem.

- Encryption: To encode, ones raises the plain text to the power'e' and deduces modulo n.
- Decryption: To decode, ones raises the crypto text to the power'd' and deduces modulon.
- The p,q are two different larger random prime (basically, having about 100 digits in their decimal representation) numbers.

2.5.1GENERATINGPRIMES

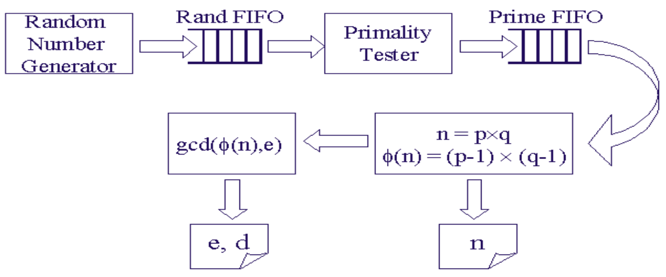


Figure 2.1: RSA key generation

Producing primes are vital to the RSA algorithmic rules. This method used here includes 2 tests for vitality. 1st a accidental odd number of the suitable size is created, by a crypto-graphically protected random value producer. 2nd , the number is proved by dividing it by 3, 5, 7, 11, 13, 17 and 19 to cross over the non-prime numbers. This method can be sustained for less values, but isn't viable for long (20 / greater values), so we go for other test. This 2nd vitality test is founded on Fermat's little theorem, that if a number p is prime, then for any $2 \leq a < p$, then $a^{p-1}=1$ (modulo p). For compound values, this procedure generally not every time it fails. So this test is oftentimes recurrent until an anticipated level of confidence is got. A value that clears all the tests is confirmed prime. Note that ideally the 2 primes should not be quite the equal size. Still around factorization algorithmic can fastly found two similarity-sized factors of a values.

2.5.2 GENERATINGENCRYPTIONANDDECRYPTIONEXPONENTS

To compute the exponentials, we must obtain 2 values: Firstly an encoding exponent e comparatively prime to (p-1)*(q-1), and then next decoded exponent d such that e * d ≡ 1 (modulo (p-1)*(q-1)). These can be done concurrently using a distinction on Euclid's algorithmic:

- Putx[0]=(p-1)*(q-1)
- Puty[0]=0
- Pu x[1]=e
- Puty[1]=1
- Then while[i] > 0 calculate:
- x[i]=x[i-2] modulo x[i-1]
- y[i]=y[i-2]-floor(x[i-2] / x[i-1])*y[i-1]
- We shown by induction that x[i]≡e * y[i](modulo (p-1)*(q-1)), so if x[i]=1 we know both that e is moderately prime to (p-1)*(q-1), and that y[i](modulo (p-1)*(q-1)) is the correspondence value for d.

2.5.3 ENCODING / DE-CRYPTION

The en-cryption equivalence m^e (modulo n) is easy, but very instance overwhelming, particularly when you comprehend that n may be 1000 or more bits longer, even earlier we initiate rising it to the power. Couple of things makes this practical: 1st is the basic power rule $x^{y+z}=x^y*x^z$. So we can determine m^e by continuously squaring m and multiplying the suitable powers togetherness. The other equivalence that aids is a*b (modulo n) ≡(a (modulo-n))*(b modulo-n))(modulo-n), so we consider the value modulo-n as often we want, generally after each multiply operation.

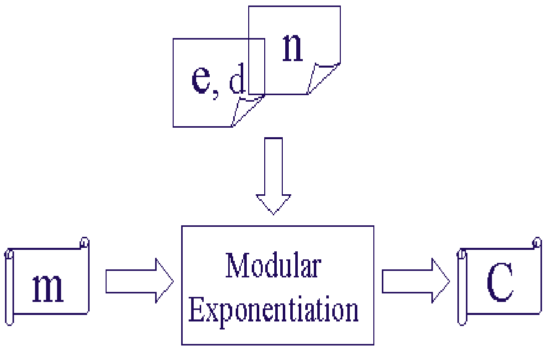


Figure 2.2 : The RSA Encryption

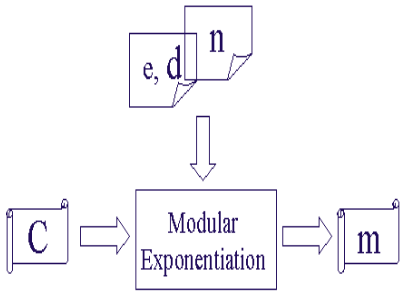


Figure 2.3: The RSA Decryption

2.5.4DECRYPTIONUSINGTHE CHINESEREMAINDERTHEOREM

The up process is yet dragging, but can be made much speeder, by manipulating our cognition of P & Q when we decode a message. We end up doing 2 standard exponentiation, but still the values(and exponents) are lesser, we finish it up as much as 4 times fastness totally. Often this is compounded with a little such as 3, 17, or 65537 to keep the encoding period sensible. The detail description of Chinese remainder theorem is given in section 2.6.5.

2.5.5 RSA EXAMPLE

T

- Choose primes:p=17&q=11
- Calculate n=p*q=17*11=187
- Calculate Φ (n)=(p-1)(q-1)=16*10=160
- Choose=gcd(e,160)=1; choose: e=7
- Find d: de=1 mod(160) ; d<160
- If d=23 so 23*7=161=10*160+1
- Put out open key: ku={7,187}
- Keep secret private key : kr={23,17,11}
- Known message M=88(M<n)
- Encode: C=88^7 mod187=11
- Decode: M=11^{23} mod187=88

2.6 DIFFERENT THEOREMS ARE USED TO COMPUTE RSA

2.6.1 GCD (GENERAL DIVISORS AND GREATEST-COMMON-DIVISOR)

For a, b ∈Z \{0}, c ∈Z is a common divisor of a&b if c/a and c/b. Furthermore, c is the greatest common divisor, denoted gcd (a,b), if it is the biggest integer that dividing a&b. Another possibility to define the highest general divisor of a & b is to say that c = gcd(a,b) if any common divisor of a&b also divides c. gcd (0, 0) = 0,and gcd (a, 0) = |a| for all a ∈Z \ {0}. If a, b ∈Z \ {0}, then 1 ≤ gcd (a, b) ≤min {|a|, |b|} and gcd (a, b)=gcd (±|a|, ±|b|). Consequently, the greatest-common divisor of two integers can never be negative (even if one or both integers are negative). Furthermore, 2 numerals a, b ∈Z \ {0} are moderately prime or co-prime if their great common divisor is 1 (i.e., if gcd(a, b)=1).Similar to the (great)common divisor, it is possible to define the (least)common multiple of 2 numerals.

- Gcd(a, b)=the biggest integer that/ a & b.
- If gcd (a,b) =1, then a&b are comparetively prime.
- Modulo operator ‘a mod n’, remainder of ‘a’ when /by n.
- ‘ a’ is matching to ‘b modn’ ,if

i. a mod n=b mod n
ii. a=kn+r, b=tn+r
iii. a-b=fn

2.6.2 EUCLIDEAN ALGORITHMS

In fore coming Section we see how one can compute the greatest-common-divisor of 2 numerals if their prime factorization is known. It is, however, not essential to know the prime factorization of 2 numerals to calculate their highest-common-divisor. In fact, the Euclidean’s, algorithm (or Euclid’s algorithm) can be used to calculate the highest-common-divisor of 2 numerals a, b ∈Z \ {0} with unknown prime factorization.

Here we are considering one theorem: it says that for 2 nonzero integers a ≥ b, we can always write a = bq + rfor some quotient q _= 0 and remainder 0 ≤ r < b. Because by explanation, gcd (a, b) dividing both a & b, the above equation shows that it must also divide r. Consequently, gcd (a,b) = gcd (b, r), and so that the remainder r of a dividedby b is written by a mod b, we can say gcd (a, b) = gcd (b, a mod b) = gcd (b, Rb (a)).This equation can be recursively applied to calculate gcd of (a, b). For ex: gcd (100,35) can be calculated as given below:

Gcd(100, 35)=gcd (35, R35 (100))
=gcd (35, 30)
=gcd (30, R30 (35))
=gcd (30, 5)
=gcd (5, R5 (30))
=gcd (5, 0

2.6.3 FERMAT’S LITTLETHEOREM

This can be turned into a simple primarily testing algorithm. Fermat’s Little Theorem proves that for any prime-number Pand every value a not / p, the equivalent $a^{p-1} \equiv 1$ (mod p) must in hold. Therefore, one can trial the primarily or instead of the composites of n by randomly selecting a value for a (not/y n) and performing a^{n-1}(mod n). If this value is not equal to one, then n is definitively not a prime. Unfortunately, the converse is not true and finding afor which $a^{n-1} \equiv 1$ (mod *n*) does not imply that *n* is prime.

The (Zp+) indicates a field for all prime p (and hence (Zp*) is a group in which each element is invertible). The below mentioned theorem applies to all elements in Zp*. It is due to Pierre de Fermat, and hence it is called as Fermat’s little-theorem. It has many applications, for example: it can be used to compute the inverse multiplication of a modulo p. If we divide the equivalence $a^{p-1}=1$(mod p) by ‘a’ on either side, we get $a^{(p-1)-1} = a^{p-2} = a^{-1}$ (mod p).This means that one can find the multiplicative inverse element of ‘a modulo p’ by calculating a^{p-2}(mod p).

Example: p=7 then the multiplicative-inverse of 2 modulo 7 can be calculated as:

$2^{-1}=2^{7-2}=2^5$(mod 7) =32 (mod 7) =4.

2.6.4 EULER’S TOTIENT FUNTION

Euler’s totient function was first proposed by Leonhard Euler; as a function that counts the numbers that are lesser than *n* and had no other frequent divisor with *n* other than 1 (i.e., they are co-prime with *n*). This mathematical function is formally described as

Follows Φ (n) = | {a ∈ {0 . . . n − 1} | gcd (a, n)=1}|

The Euler’s totient function has the below properties:

- If p is prime’s , then each number lesser than p is co-prime with p. Consequently, this equation $\Phi(p) = p - 1$ holds for every prime number.
- If *p* is prime’s and $1 \le k \in Z$, then $\Phi(p^k) = p^k – p^k/p$. This is because every p th number between 1 and $p^k$is not co-prime with p^k(because p is a common divisor of pi (for i = 1 . . . k−1) and p^k) and we have to subtract p^k/p from p^k accordingly. Note that $p^k–p^k/p = p^k–p^{k-1}= p^{k-1} (p - 1)$, and hence $\Phi(p^k) = (p - 1) p^{k-1}$(this is the equation that is often found in textbooks).
- If n is the product of two primes p and q (i.e., n = pq), then $\Phi(n) = \Phi(p)\,\Phi(q) = (p - 1)(q - 1)$. This is because the numbers 0, p, 2p,..(q − 1) p, q, 2q. . (p− 1)q are no co-prime with related to n, and there are 1+(q − 1)+(p−1)=p+q−1 of these numbers (they are all different from each other ifp = q). Accordingly, $\Phi(n)=pq-(p+q-1)=pq-p-q+1=(p-1)(q-1)$.

Then putting the results together, we can determine Φ (n) for any integer *n* that has a known prime factorization (i.e., $n=\prod_i q_i^{ki}$) $\mathbf{\Phi(n) =\prod_i q(i-1)q_i^{k(i-1)}}$.

For ex, in order to perform Φ (45)we should find the factorization $45=3^2.5$ and applying the formula granted previously. In fact, we have

$\Phi(45) = (3 - 1) \cdot 3^2 - 1 \cdot (5 - 1)^{1-1}$

$= 2\cdot 3^1 \cdot 4 \cdot 5^0$

$= 2\cdot 3 \cdot 4 \cdot 1$

= 24.

A last word is due to the trouble of computation Φ (n) as related to find the factorization of *n*. For simplicity, we assume *n* to be the multiply of 2 primes p&q that is, n=pq. In this cause, we can display that calculating Φ (n) is evenly hard to find p & q. This means that if we can compute Φ (n) for any n, then we can also factor n.

2.6.5 CHINESE REMAINDER THEOREM

If $x \equiv a_1$(modulo n_1), $x \equiv a_2$(modulo n_2). .., $x \equiv a_k$(mod n_k) be a system of k congruence’s with pair wise co-prime moduli $n_1 . . . n_k$. The system has a unique and efficiently computable solution x in Zn with $n =\prod_{i=1}^{k} n_i$.The fact that the solution is unique in Zn means that all other solutions are not elements of Zn. In fact, the set of all results is the set of integers y such that y≡x (mod n). Furthermore, the fact that there is an efficiently computable solution x in Zn means that there is an efficient algorithm that finds the solution. This algorithm is sometimes referred to as the Chinese remainder algorithm (CRA).

Let m_i= n/$n_i$for i = 1. . . k, and v_i= m_i^{-1}(mod n_i), meaning that v_i is the multiplicative inverse element of m_i modulo n_i. Because all modules are assumed to be pair wise co-prime, v_i is well defined (for all i= 1…., k). The solution x is computed as follows: x=$\sum_{i=0}^{k}$ ai mi vi (mod n)

For example, consider the following system of 3 congruence's:

x≡5 (mod 7)

x≡3 (mod 11)

x≡11 (mod 13)

In this example, n_1= 7, n_2= 11, n_3= 13 (note that these integers are pairwise co-prime), and n = 7· 11 · 13 = 1001. Furthermore, a_1= 5, a_2= 3, and a_3= 10. To find the ressult x in Z_{1001}, we should perform:

m_1= 1001/7 = 143

m_2= 1001/11 = 91

m_3= 1001/13 = 77 and

y_1≡ 143−1 (mod 1001) = 5

y_2≡ 91−1 (mod 1001) = 4

y_3≡ 77−1 (mod 1001) = 12

From this, the result x can be computed as given below:

x≡$\sum_{i=0}^{k}$ ai mi vi (mod n)

≡$a_1m_1y_1$+ a_2 m_2 y_2+ $a_3m_3y_3$(mod n)

≡5 · 143 · 5+ 3 · 91 · 4 + 10 · 77 · 12 (mod 1001)

≡3575 + 1092 + 9240 (mod 1001)

≡13907 (mod 1001)

= 894

Accordingly, x = 894 is the result in Z1001, and {i ∈Z | 894 + i·1001} is the set of all results in Z.The case k = 2 is so important in practice that we have a closer look at the corresponding CRA. The system of congruence's looks as given below:

x≡a_1(mod n_1)

X=a_2(mod n_2);

Again, the moduli $n_1$and $n_2$must be co prime (is., gcd (n_1, n_2) = 1). We can compute t ≡ n_2^{-1}(mod n_1) andu ≡(a_2–a_1) t (mod n_1). The answer x modulo n can then compute as follows:x= a_1+ u n_2.The CRA can be used to speed up the execution of many open key crypto-systems, including the RSA open key crypto-system.

3. Implementation Of L.M.Montgomery Standard Multiplication Algorithm

The standard multiply operation algorithmic that authenticate the process of standard multiplication generally comprise of 2 steps: 1. produces the creation P=A×B and the another contracts this result of P modulo M. The direct way to calculate a product is built on an repetitive add-on-accumulator for the produced partial effect. But, this solutions often lag as the end outcome is only attainable after n clk -cycles; is the size of the variables. A quicker form of the repetitive multiply shall addon many partial products at one time. This could be obtained by recounting the repetitive multiply and yields a combinational circuit that includes several partial product initiators along with many adders that work in simultaneously. A commonly used algorithm for proficient standard multiply operation is the L.M.Montgometry's algorithmic. It performs the multiply of 2 variables modulo a third one without executing division by M using this algorithmic rules. It yield the deduced product using a order of add-ons. Let A, B & M be the parameters of multiplication and the modulo respectively and let n be the no of values in their binary demonstration that is the radix is 2. So, we noted A, B& M as given below:

$$A=\sum_{i=0}^{n-1} a_i \times 2^i, \quad B=\sum_{i=0}^{n-1} b_i \times 2^i \text{ and } M=\sum_{i=0}^{n-1} m_i \times 2^i$$

The Standard Multiply operation is the numerical operation on variables X*Y mod M with X,Y< M, thereby X & Y are the variables and M is modulus. In recent applicable crypto usages X,Y& M are big variables. There are several various algorithm’s for standard multiply operations. In this work we are focusing on the 2 most significant one’s. During interspere standard Multiply opeartion and the computing of the remainder of the division are furnished. The benefit is that the duration of the intermediary outcome is only one or 2 bits bigger than the operands. The drawback is the use of deductions in order to diminish the intermediate outcomes. To overcome this disadvantage we use the L.M.Montgometry standard multiply operation. The calculation is done in the L.M.Montgometry field. The gain of this computation is that we do not require subtractions in order to deduce the in-between results. The shortcoming is the fact that the L.M.Montgometry’s standard multiplication performs $X*Y2^{-n}$modM instead of X*Y mod M and 2 L.M.Montgomery’s standard multiplication’s are required for 1 standard multiplication. The key benefit of the L.M.Montgomery multiply operation contrary to another standard multiplication algorithmic is that with easy mathematical dealings (like addition and multiplication) and low processing cost, it can give accurate outcomes. The necessity for division or even reversal is bring forth in a shift action. So, many applications of the algorithm have been projected, for hardware or software. Some use full radix representations at the cost of excess complexness while others re-arrange or modifying the algorithms so as to better the execution.

The initial conditions of the L.MMontgomery algorithms are as shown below: The modulus-M wants to be comparatively prime to the radix, that is. There be no mutual divisor for M & radix. The parameters of multiplication essential to be littler than M.A s here using the binary notation of the values, then the modulus M want to be odd to satisfy the 1st initial condition. The L.M.Montgometry algorithmic uses the least significant bit of the hive away standard partial product to define the multiples of M to deduct. The normal product order is inverted by selecting multiplier integer from lowest to to the highest degree need and shifting bottom. If *R* is the present standard partial output, then q is selected such that R+q×M is a multiples of the radix *r*, & this is precise changing by *r p*osition, that is divided by *r* for use in the coming repetition. So, after *n* repetitions, the outcome obtained is $R=A\times B\times r^{-n}$mod M. A altered model of L.M.Montgometry algorithmic is shown in following algorithmic.

```
**Algorithmic -1**Montgomery (A,B,M)
**int**R = 0;
1) **for** i= 0 **to** n-1
2) R = R + $a_i$×B;
3)  **if** $r_0$ = 0 **then**
4)  R = R div 2
5)  **else**
6 R = (R + M) div 2;
return R;
**end** Montgomery.
```

We require an extra Montgometry modular multiply operation by the constant 2^n mod M In order to attain the correct outcome. Even so as the chief purpose of the use of L.MMontgomery standard multiplication algorithmic rule is to perform exponentiation, it is recommended to L.M.Montgometry inital-multiplying the operands by $2^{2n}$and Montgometry next-multiplying the result by one to get residue of the 2^{-n} term. Here we contemplate on the implementation of the Montgometry multiplication algorithm. To obtain the correct outcome, we want an excess L.MMontgomery standard multiplication by the constant r^{2n} mod M. Here we are using binary creation of numbers, we calculate the last outcome using the following algorithmic rule.

```
**Algorithmic rule_2**Modular Multiplication (A,
B, M, n)
**Constant** C: = $2^n$ modM;
**int**R: = 0;
R: = Montgometry (A, B, M);
**Return** Montgomery(R, C, M);
**end** Standard Multiplication.
```

3.1 MODULARMULTIPLIER DESIGN

The standard multiplier attains the definite value of A×B modM. Iinitially computes R= $A\times B\times 2^{-n}$modM by the use of the L.MMontgometry standard multiplier. Then, it performs R × C modM, where C = 2^nmodM.

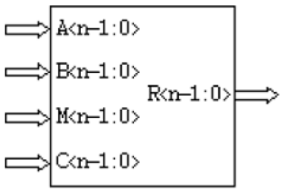


Figure 3.1: The standard multiply interface

• Step-1: It gives R1= A×B×2^{-n}mod M. The reg R1 having this result.

• Step-2: It gives R= R1×C then R contains the result of modular multiplication algorithm (where C=2^n mod M).

3.2 REPETITIVE MONTGOMETRY DESIGN

In this segment, we summarize the study of the Montgometry standard multiply. The network of the L. M.Montgometry standard multiply is in Fig 5. It takes the values A,B & M and it calculates R = (A×B×2^{-n}) mod M.The brief design of the Montgometry standard multiplier is given in Fig 26. It uses couple of muxes, 2 adders, 2 shif- registers, 3 register's and a controller.

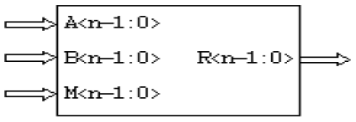


Figure 3.2: Montgomery multiplier interface

The 1st mux of the planned architectures, is. MUX-2_1 allows 0 or the contents of reg B depends on either bit a0 means 0 / 1 severally. The 2nd mux, is. MUX-2_2 allows 0 / the contents of reg M depends on either bit r0 means 0 / 1 correspondingly. The pre add-on, i.e. ADDER-1, yields the addition of R + a_i× B (line 2 of algorithmic of Fig. 1), and the 2nd adder, i.e. ADDER-2, yields the addition of R+M (line-6 of the similar algorithmic rule).

The SHIFT-REGISTER-1 delivers the bit a_i. In each repetition i of the reproducer, that shift-reg is shifted to right one time so that a0 having a_i. The task of the organizer belongs of synchronize the shifting and loading operation's of the SHIFT-REGISTER-1 and SHIFT-REGISTER-2. It also power to the many values of repetitions that have to be calculated by the multiply operation. So, the controller uses a normal bottom counter. The counter is essential to the regulator.

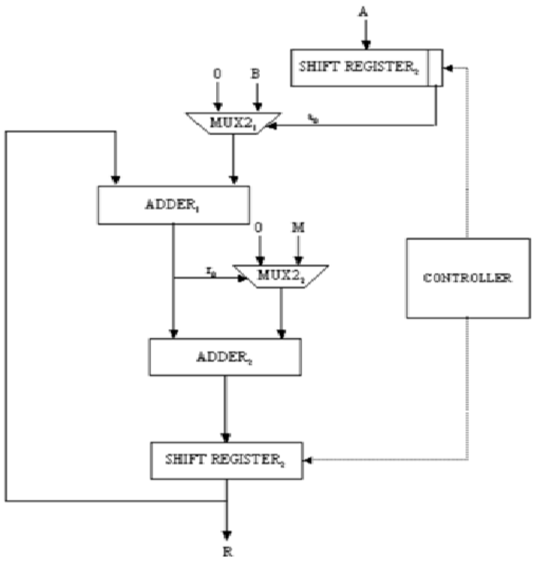


Figure 3.3: Iterative Montgomery modular Multiplier

The Architecture takes the operand's A,B & M and performs R = (A×B×2^{-n}) modM.

- MUX2_1 = 0 when bit a_0 = 0

= B when bit $a_{0 = 1}$

- MUX2_2 = 0 when bit r_0 = 0
 = M when bit $r_{0 = 1}$

- ADDER1 o/p = (R+ai× B) (line 2 Algorithmic_1)
- ADDER2 o/p = (R+ M) (line 5 of the Algorithmic_1).

In order to coordinate the process of the elements of the design, the organizer belonging to a state machine, this has six steps and is described below:

• State-0: Set the state machine; soGo to State-1;

• State-1: Loading multipliplier and modulus into their related locations. Loading multiplier into the shift register1; Go to State-2;

• State-2: Remain for ADDER1; Wait for ADDER2; Loads multiplier into shift register2;Increase Counter; Go to State-3;

• State-3: Alter shift register2; Alter shift register1; Go to State-4;

• State-4: Checking the counter; If 0 then go to S5 else go to State-2;

• State-5: Stop.

3.3 SYSTOLIC MONTGOMERY DESIGN

An improved version of Montgomery algorithm_3is shown below. The last significant bit's ofR + a_i×B is the last importance bit's of the addition of the last needed bit's of R and B ifa $_i$is 1 and the LSB of Relse. Also, latest ranges of R are either the older ones added up with a_i×B or with a_i×B + q_i×M depends onwhether $q_i$is 0 / 1.

```
**Algorithmic_3** Modified Montgomery (A,B,M)
int R:= 0;
1)  for i:=0 to n-1
2)  $q_i$:= (r0 + $a_i$×$b_0$) mod 2;
3)  R:= (R + $a_i$×B + $q_i$×M) div 2;
return R;
end
 Changed Montgometry.
```

The processing elements are performs 4-bits of a standard multiplication. The processing elements, standard multiplication and modular exponentiation have a close relationship.

The relationship is explained with the following definitions:

- Processing element: It performs 4 bit's of a standard multiply operation.
- Standard multiply operation: An collection of developing elements performs a standard multiply operation's.
- Standard exponentiation: It mixed the standard multiplication to a standard exponentiation's.

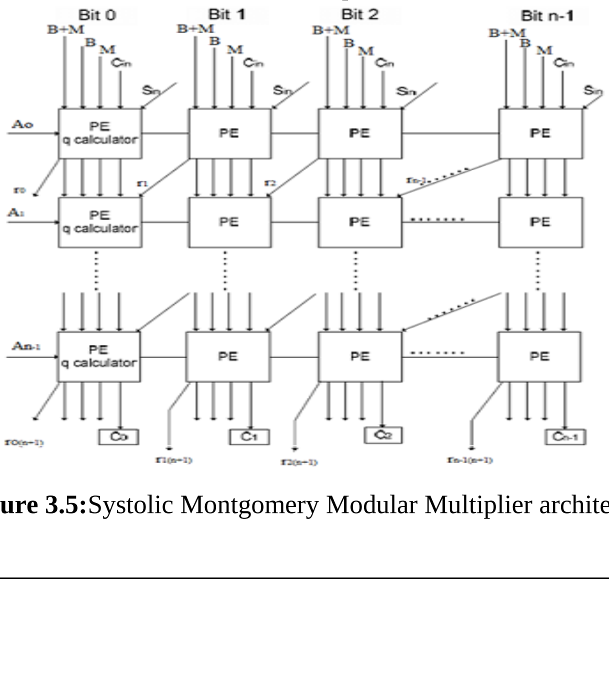


Figure 3.5: Systolic Montgomery Modular Multiplier architecture

The Montgomery systolic architecture functioning is mainly depends on the processing elements (PE's).The processing element is generally a computation of 4-bits; the purpose of 4-bits indicates the four different operations. The 4-bits are 0, 1, 2, 3; these all are indicated as 2-bit binary values that are 00,01,10,11. In this two bits one is denoted as' a_i' and another bit is 'qi', the following table gives the computation of each bit.

ai	qi	R + aixB + qixM
1	1	R + MB
1	0	R + B
0	1	R + M
0	0	R

Table 3.1:calculation of $(R+a_i \times B+ q_i \times M)$

In the table 3.1, the result depends on the two bits a_i, qi .To get the total result the a_i will be multiplied by the 'B' and q_i will be multiplied by the 'M' and these two products added with the previous result, then we can get the total result as $R +a_i \times B+ q_i \times M$.

- If the a_i,q_i bits are 0, 0 then the result is same as previous row result; there is no change in result.
- If the a_i,q_i bits are 0, 1 then the result is R+M: previous result added with the modulus value (M).
- If the a_i,q_i bits are 1, 0 then the result is R+B: previous result added with the multiplier value (B).
- If the a_i,q_i bits are 1, 1 the result is R+MB:previous result added with the multiplier, modulus values.

The processing element architecture designing will be explained by using the following Systolic Montgomery algorithm:

```
Algorithm_4 Systolic Montgomery (A,B,M,MB)
Int  R: = 0;
bit carry: = 0, x;
0)  for i: = 0 to n
1)  qi: = $r_0^{(i)} \oplus a_i.b_0$;
2)  for j: = 0 to n
3)  switch $a_i, q_i$
4)  1, 1: x := $mb_i$;
5)  1, 0: x := $b_i$;
6)  0, 1: x: = $m_i$;
7)  0, 0: x := 0;
8)  sum:  $r_j^{(i+1)} = r_{j+1}^{(i)} \oplus x_i \oplus$ carry;
9) carry:= $r_{j+1} . x_i + r_{j+1}$.Carry+xi.carry;
return R;
end Systolic L. M. Montgomery.
```

- Every PE had to perform 5 to 3 additions using present value of q_i &a_i.
- The first PE has an additional responsibility to compute the value of quotient $q_i = ((r_0 + a_i \times b_0) \bmod 2)$.
- At each iteration of algorithmic_4, line 4 presenting the q_i quotients calculation so that $(R +a_i \times B+ q_i \times M)$ become's a multiples of 2.

The discipline of the basic PE,is $cell_{i,j} 1 \le i \le n-1$ & $1 \le i \le n-1$, is given in picture 3.6; in PE architecture one multiplexer and one full adder are there. The multiplexer inputs are $0,m_i,b_i,mb_i$ (from algorithm_4) and selection lines are a_i,q_i. The output of multiplexer given to the full-adder. The output of the full-adder is given to the next processing element.

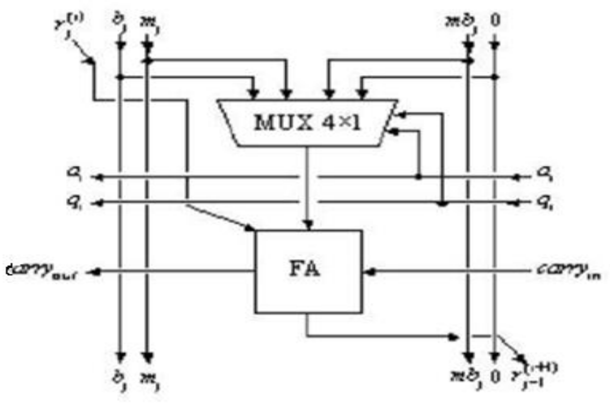


Figure 3.6: Internal architecture of general PE

General PE is doing only addition: Sum_{i+1} = $Sum_i T_i$+T_ix$Carry_i$+$Carry_i$xSum

T_i = 0 when a=0, q = 0

= M when a=0, q = 1

= B when a=1, q=0

= M+B when a=1, q=1

Itapplies the instructions of line's 2-9 in systolic L. M. Montgomery algorithmic rule_4. The architecture ofthe next-most highest PE,is. Cell 0,0, executes the calculation of lines2-9, it implementsthe calculation indicates in line 1. Still as $r_0^{(0)}$ is zero, the calculation of q_0 is decreased to $a_0.b_0$. Also, the full adder is not needed as carry in signal is also 0. So $r_1^{(0)}\oplus x_i \oplus$carry & $r_1^{(0)}.x_i$+ $r_1^{(0)}$.carry+x_i.carry are deduced to x_i&0.

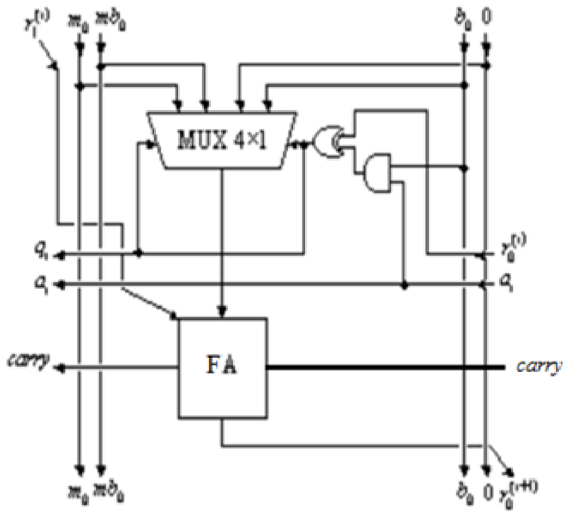


Fig 3.7: Internal architecture of quotient PE

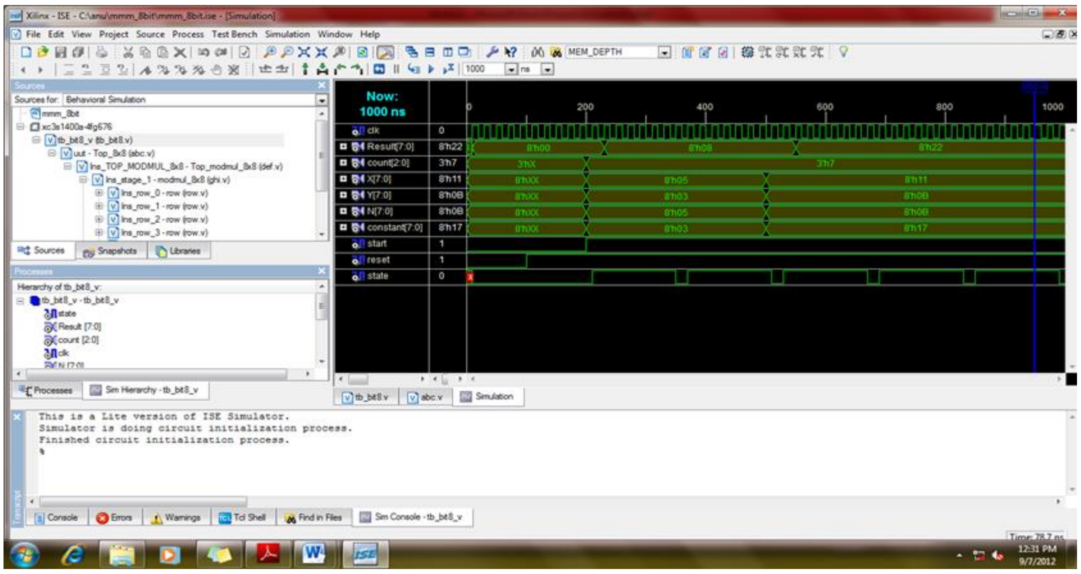
The internal architecture of the quotient block used to compute iteration of Algorithm_4, line 4 presents the qi quotient computation so that (R+a_i×B+ q_i ×M) becomes a multiple of 2.

The architectures of general processing element and quotient processing element are same but the quotient architecture has some extra circuit to calculate q_i. The first PE (that is quotient processing element) has an additional responsibility to compute the value of quotient q i = ((r_0 + a_i×b_0) mod 2).

It implements the first line of the algorithm_4.It executes the calculation showed in line1 of algorithm_4. Still as $r_0^{(0)}$ is 0, the calculation of q_0 is decreased to $a_0.b_0$. In quotient PE architecture one multiplexer and one full adder are there. The multiplexer inputs are 0,m_0,b_0,mb_0 (from algorithm_4) and selection lines are a_i,q_i. The output of multiplexer given to the full-adder. The output of the full-adder is given to the next processing element.

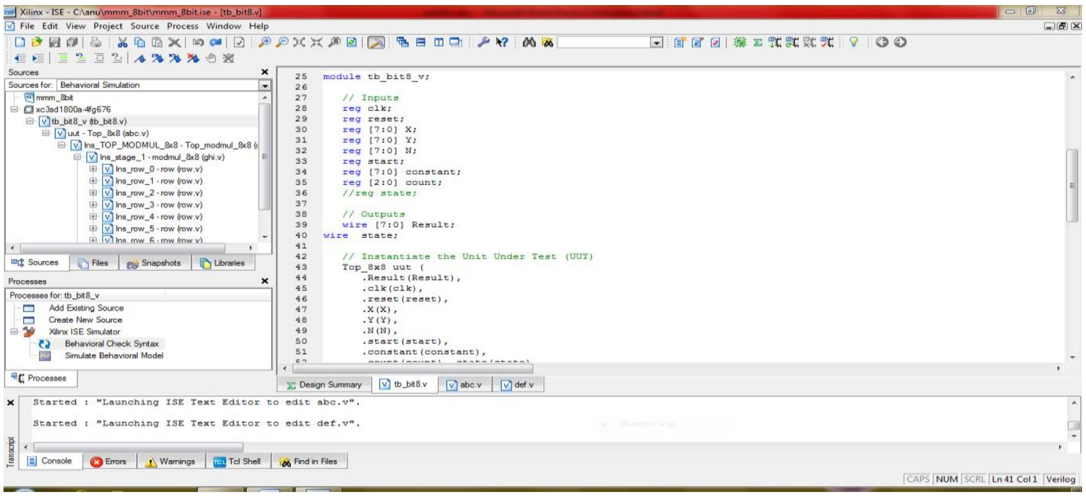
4. Results

SIMULATION RESULTS AND SYNTHESIS REPORT



```
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
%
```

Figure 4.1: 8-bit operand result for (X, Y, N) = (5, 3, 5)



```
module tb_bit8_v;

   // Inputs
   reg clk;
   reg reset;
   reg [7:0] X;
   reg [7:0] Y;
   reg [7:0] N;
   reg start;
   reg [7:0] constant;
   reg [2:0] count;
   //reg state;

   // Outputs
   wire [7:0] Result;
wire  state;

   // Instantiate the Unit Under Test (UUT)
   Top_8x8 uut (
      .Result(Result),
      .clk(clk),
      .reset(reset),
      .X(X),
      .Y(Y),
      .N(N),
      .start(start),
      .constant(constant),
```

```
Started : "Launching ISE Text Editor to edit abc.v".

Started : "Launching ISE Text Editor to edit def.v".
```

Figure 4.2: Test bench module for 16-bit operand

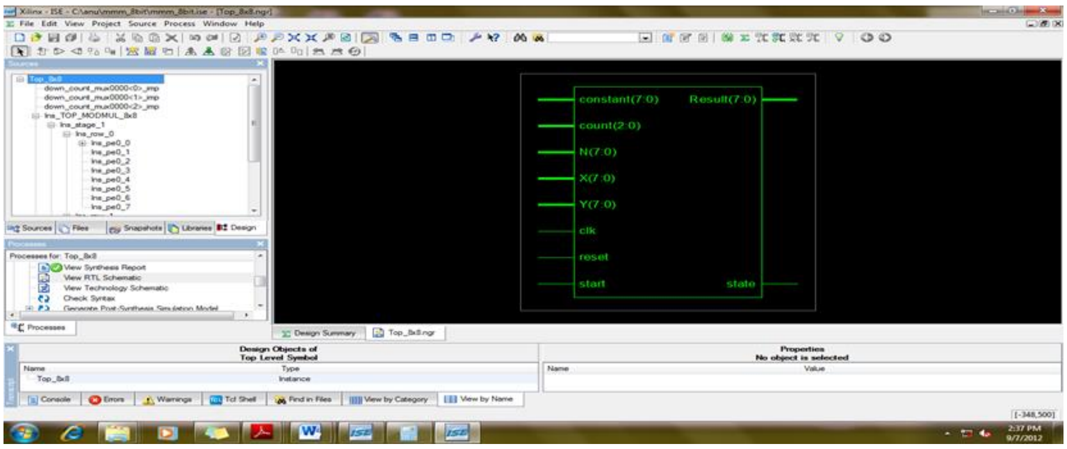


Figure 4.3: 8-bit operand top module

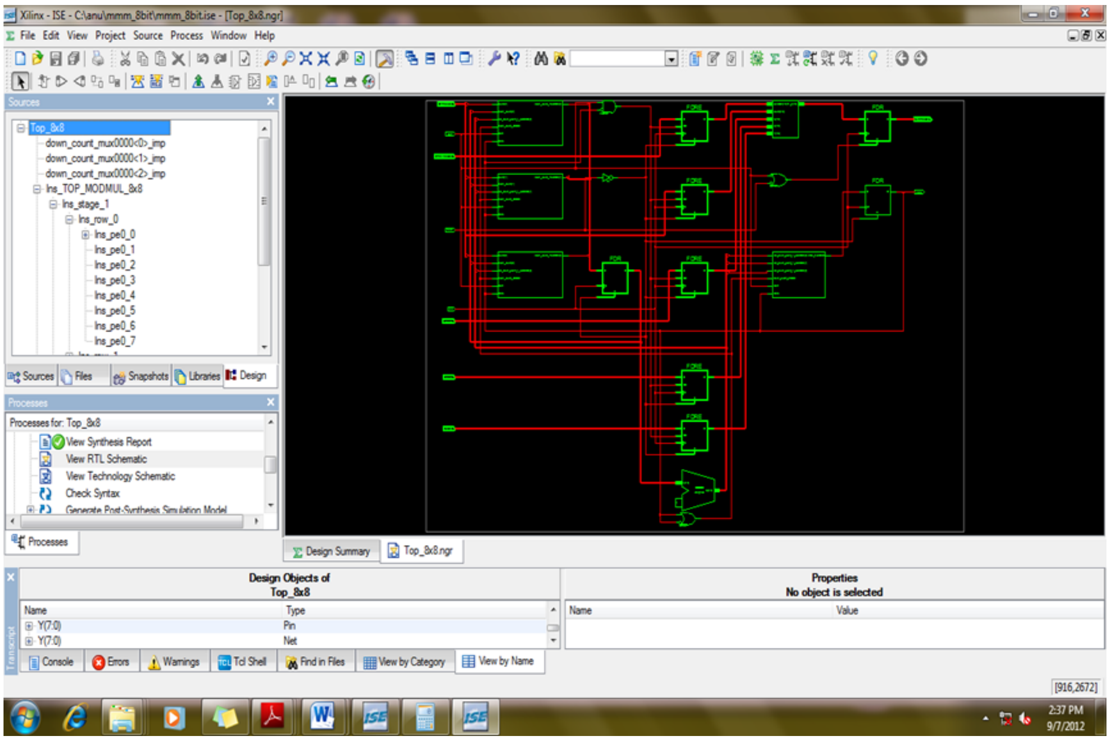
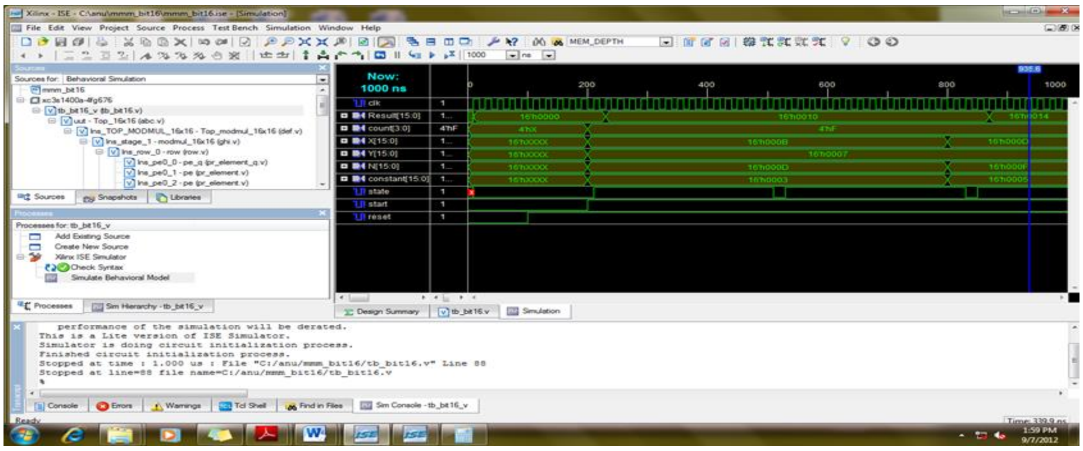
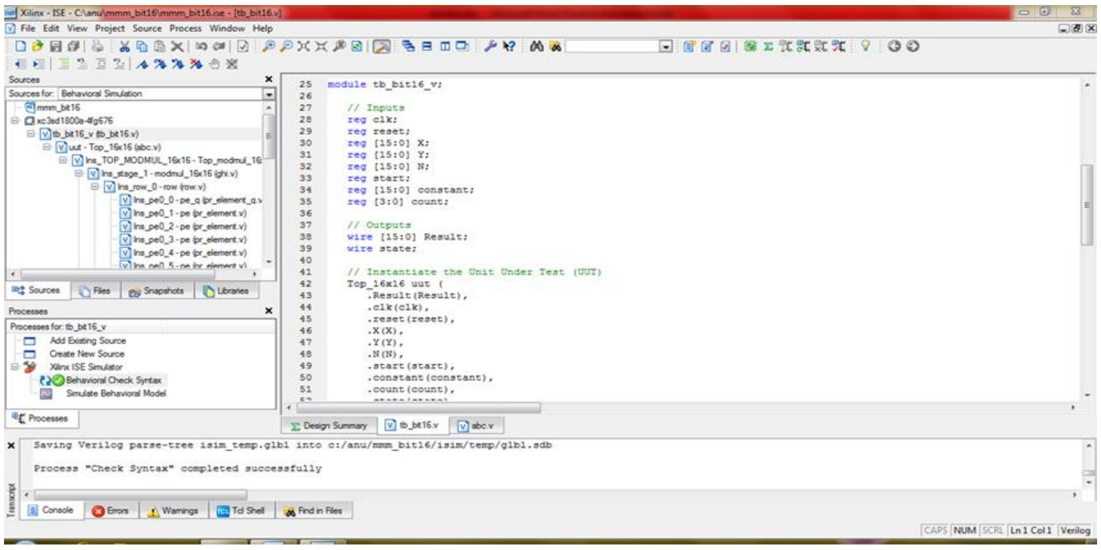


Figure 4.4: 8-bit operand RTL schematic



```
   performance of the simulation will be derated.
This is a Lite version of ISE Simulator.
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 1.000 us : File "C:/anu/mmm_bit16/tb_bit16.v" Line 88
Stopped at line=88 file name=C:/anu/mmm_bit16/tb_bit16.v
%
```

Figure 4.5: 16-bit operand result for (X, Y, N) = (11, 7, 13)


Xilinx - ISE - C:\anu\mmm_bit16\mmm_bit16.ise - [tb_bit16.v]
Processes for: tb_bit16_v
Add Existing Source
Create New Source
Xilinx ISE Simulator
Behavioral Check Syntax
Simulate Behavioral Model

```
module tb_bit16_v;

    // Inputs
    reg clk;
    reg reset;
    reg [15:0] X;
    reg [15:0] Y;
    reg [15:0] N;
    reg start;
    reg [15:0] constant;
    reg [3:0] count;

    // Outputs
    wire [15:0] Result;
    wire state;

    // Instantiate the Unit Under Test (UUT)
    Top_16x16 uut (
        .Result(Result),
        .clk(clk),
        .reset(reset),
        .X(X),
        .Y(Y),
        .N(N),
        .start(start),
        .constant(constant),
        .count(count),
```

```
Saving Verilog parse-tree isim_temp.glbl into c:/anu/mmm_bit16/isim/temp/glbl.sdb

Process "Check Syntax" completed successfully
```

Figure 4.6: Test bench module for 16-bit operand

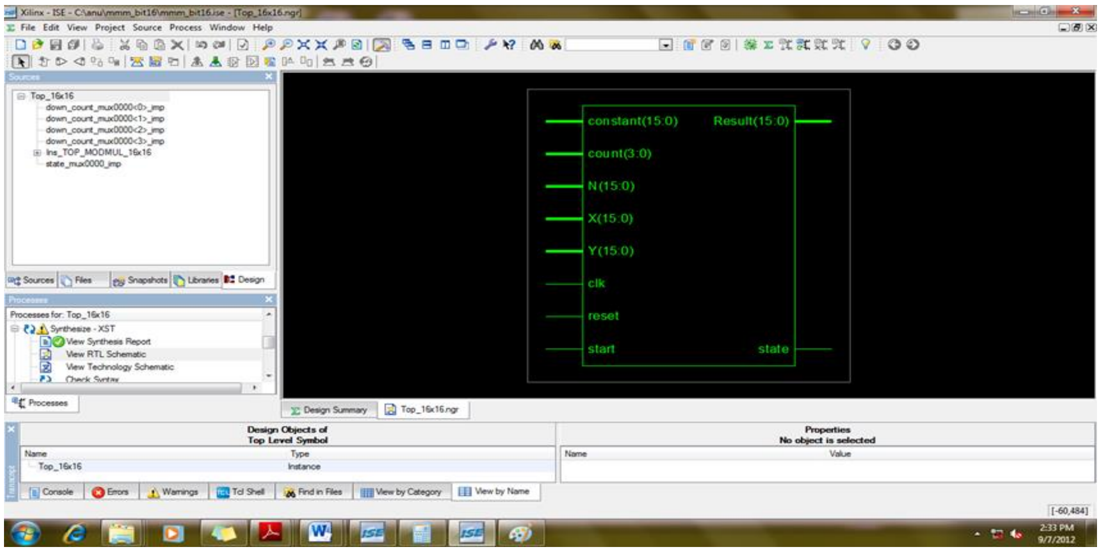


Figure 4.7: 16-bit operand top module

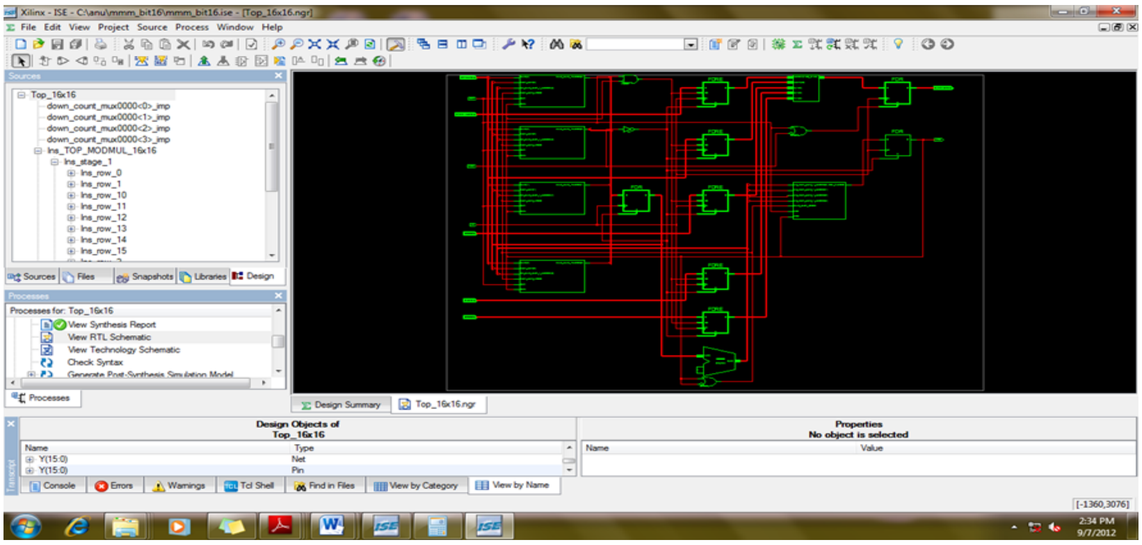
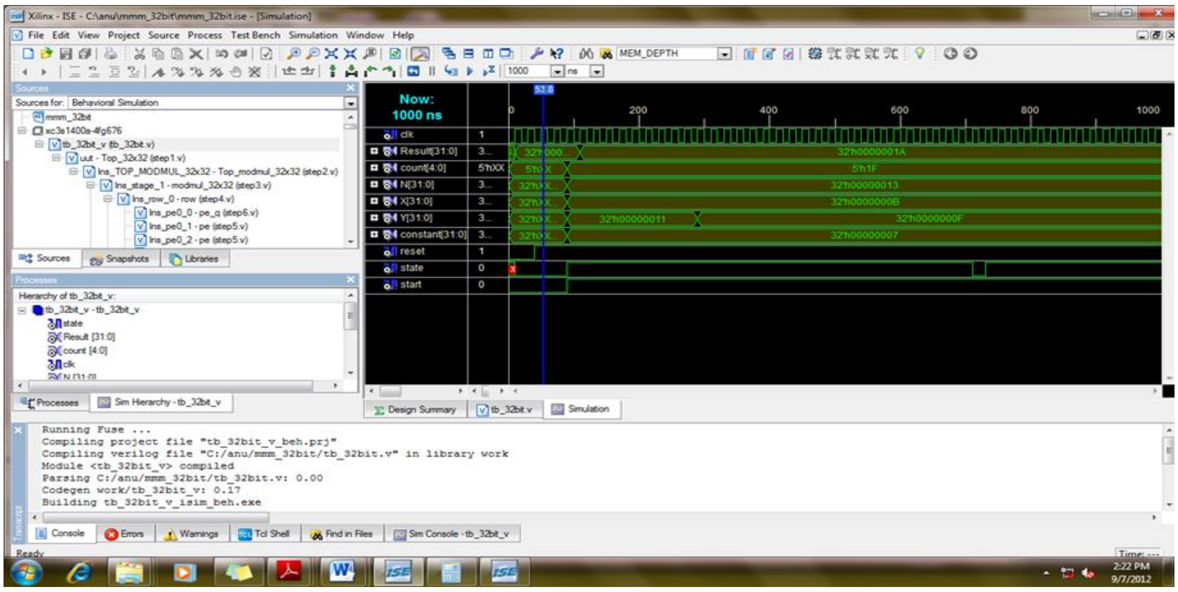

Xilinx - ISE - C:\anu\mmm_bit16\mmm_bit16.ise - [Top_16x16.ngr]
Top_16x16
down_count_mux0000<0>_imp
down_count_mux0000<1>_imp
down_count_mux0000<2>_imp
down_count_mux0000<3>_imp
Ins_TOP_MODMUL_16x16
Ins_stage_1
Ins_row_0
Ins_row_1
Ins_row_10
Ins_row_11
Ins_row_12
Ins_row_13
Ins_row_14
Ins_row_15
Processes for: Top_16x16
View Synthesis Report
View RTL Schematic
View Technology Schematic
Check Syntax
Generate Post-Synthesis Simulation Model
Design Summary
Top_16x16.ngr
Design Objects of Top_16x16
Properties No object is selected
Name
Type
Y(15:0)
Net
Pin
Value

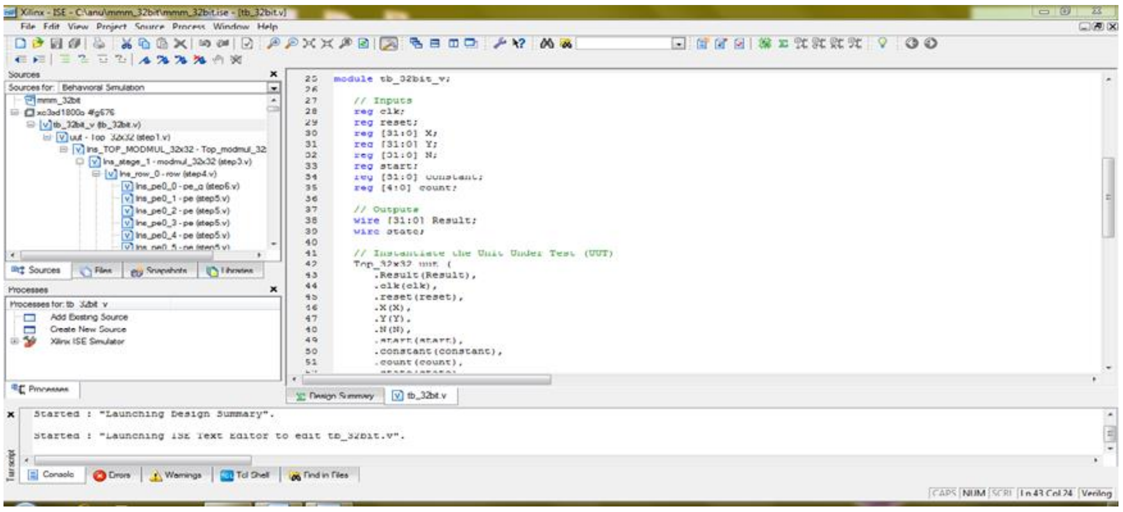
Figure 4.8: 16-bit operand RTL schematic

The test bench, top module, RTL schematic were shown in above figures 4.6,4.7,4.8. for 16-bit operand



```
Running Fuse ...
Compiling project file "tb_32bit_v_beh.prj"
Compiling verilog file "C:/anu/mmm_32bit/tb_32bit.v" in library work
Module <tb_32bit_v> compiled
Parsing C:/anu/mmm_32bit/tb_32bit.v: 0.00
Codegen work/tb_32bit_v: 0.17
Building tb_32bit_v_isim_beh.exe
```

Figure 4.9: 32-bit operand result for (X, Y, N) = (11, 17, 19)



```
module tb_32bit_v;

    // Inputs
    reg clk;
    reg reset;
    reg [31:0] X;
    reg [31:0] Y;
    reg [31:0] N;
    reg start;
    reg [31:0] constant;
    reg [4:0] count;

    // Outputs
    wire [31:0] Result;
    wire state;

    // Instantiate the Unit Under Test (UUT)
    Top_32x32 uut (
        .Result(Result),
        .clk(clk),
        .reset(reset),
        .X(X),
        .Y(Y),
        .N(N),
        .start(start),
        .constant(constant),
        .count(count),
```

```
Started : "Launching Design Summary".

Started : "Launching ISE Text Editor to edit tb_32bit.v".
```

Figure 4.10: Test bench for 32-bit operand

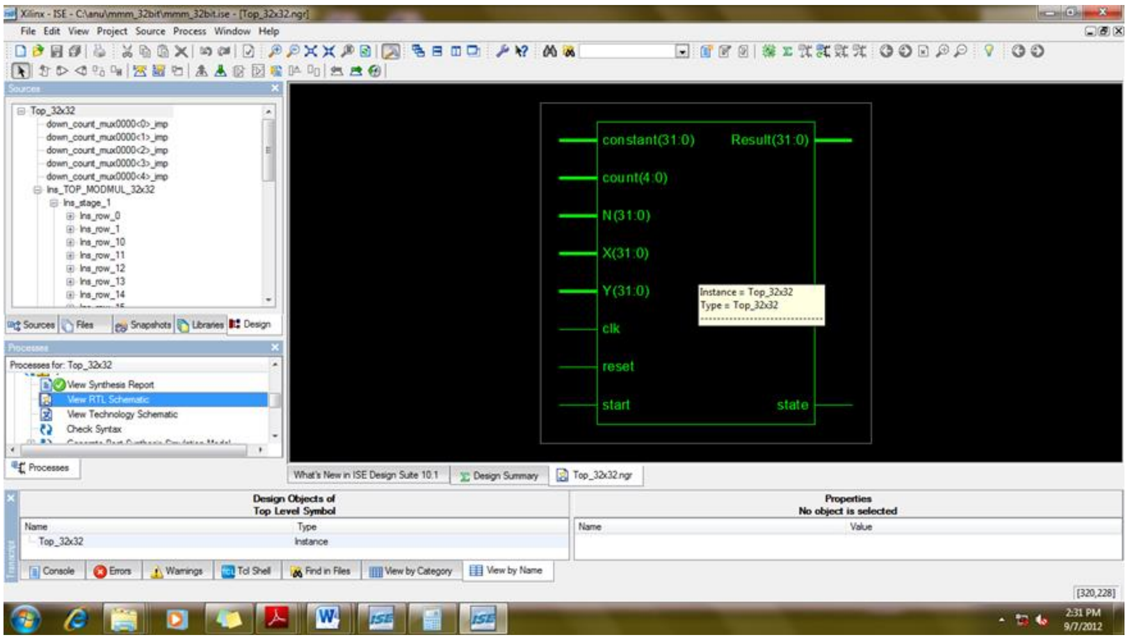


Figure 4.11: 32-bit operand top module

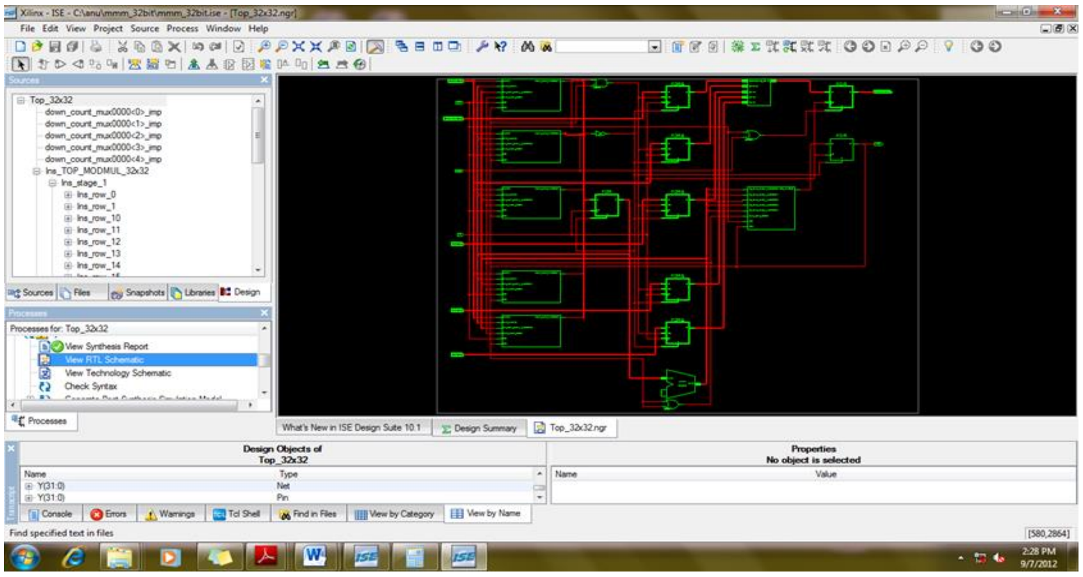


Figure 4.12: 32-bit operand RTL schematic

Similarly, The test bench, top module, RTL schematic were shown in above figures 4.10,4.11,4.12. for 32-bit operand.

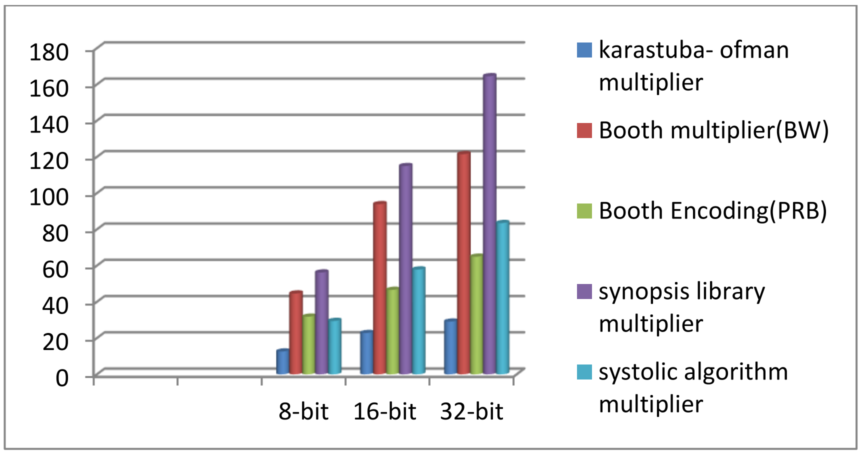
SYNTHESIS REPORT FOR 8-BIT OPERAND:

The synthesis report comparison for 8,16,32-bit operand is shown below:

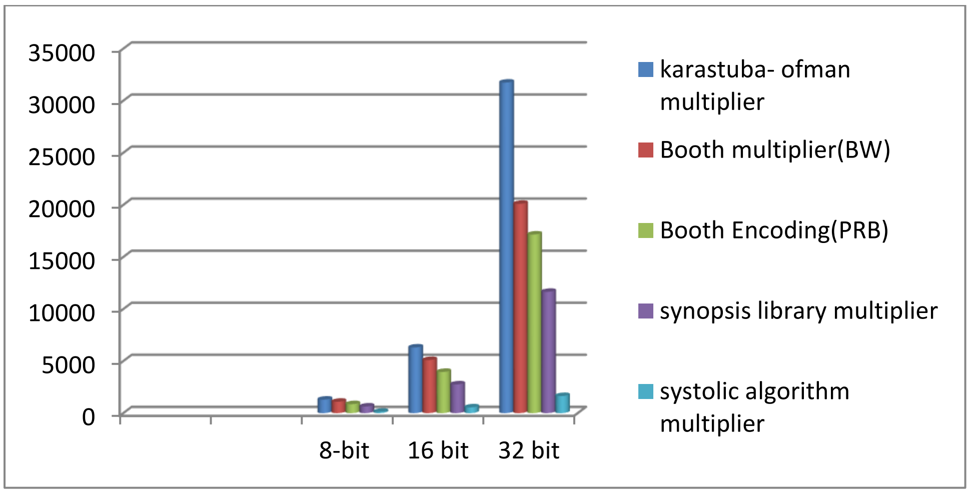
No of bits	**No of inputs**	**No of 4 input LUTs**	**delay**	**Area**
8-bit	47	982	29.472ns	128
16-bit	88	4302	57.855ns	553
32-bit	169	12770	83.484ns	1639

PERFORMANCE FIGURE OF THE SYSTOLIC MULTIPLIER ALGORITHM COMPARISON OF ALL THE PRESENTED ALGORITHMS:

The performance figure of the systolic multiplier algorithm comparison of all the presented algorithms in terms of Delay:



The performance figure of the systolic multiplier algorithm compare to all the existing algorithms in terms of Area:



5. Conclusion And Future Scope

In this project I have compared best and fresh methods for well-organized standard multiply operation. I showed that the standard multiply operation A×B mod M can be calculated in mainly by couple of ways: one is calculating the product then deducing it; or obtaining the deduced product directly. Most efficient method for multiply operation are karatsuba, offmans Booths. Are for deduction and L.M.Maontgometrys algorithmics for in-between multiply and deduction. I have detailed the Montgomery's algorithm for modular multiplication and reduction. The two firmware calculations: one iterative and the other systolic are given better performance . In this thesis, Montgomery modular multiplication respective architectures for evaluating standard multiplication have been done, implemented, simulated, and synthesized for a Xilinx FPGA.

The systolic implementation is much better than the sequential one and it requires less firmware space. Subsequently, I reviewed that current systolic technique should allows further improvement to the implementation of the standard multiplication with big operands.

References

1. Baki, A. (2008). *Kuramdan uygulamaya matematik eğitimi*. Ankara: Harf Eğitim Yayıncılığı
2. Başol, G., Balgalmış, E., Karlı, M. G., & Öz, F. B. (2016) TEOG sınavı matematik sorularının MEB kazanımlarına, TIMSS seviyelerine ve yenilenen Bloom Taksonomisine göre incelenmesi. *Journal of Human Sciences, 13*(3), 5945-5967.
3. Breakwell, G. M., Wright, D. B., & Smith, J. A. (2012). *Research questions and planning research*. Londra: SAGE Publications.
4. Businskas, A. M. (2008). *Conversations about connections:How secondary mathematics teachers conceptualize and contend with mathematical connections* (doctoral dissertation). Simon Fraser University, Canada. Çepni, S.(2014). *Araştırma ve proje çalışmalarına giriş* (7.baskı). Trabzon: Celepler Matbaacılık
5. Francisco, J. M., & Maher, C. A. (2005). Conditions for promoting reasoning in problem solving: Insights from a longitudinal study. *Journal of Mathematical Behavior, 24*, 361–372.
6. Generazzo, S. D. (2011). *Proof and reasoning in an inquiry-oriented class: The impact of classroom discourse* (doctoral dissertation) University of New Hampshire, New Hampshire.
7. Goswami U. (2004). Neuroscience and education. *British Journal of Educational Psychology, 74,* 1–14
8. Güven B., Öztürk T. ve Demir E. (2014, Eylül). *Ortaöğretim matematik öğretmen adaylarının ispat sürecindeki muhakeme hatalarının incelenmesi*. XI. Ulusal Fen ve Matematik Eğitimi Kongresi'nde sunulan bildiri, Çukurova Üniversitesi, Adana, Türkiye.
9. Güven B. ve Demir E. (2015, Mayıs). *Öğrencilerin İspat Sürecinde Yaptıkları Muhakame Hatalarına Yönelik Öğretmen Bilgisinin İncelenmesi*. 2. Türk Bilgisayar ve Matematik Eğitimi Sempozyumu'nda sunulan bildiri,
10. Adıyaman Üniversitesi, Adıyaman, Türkiye.
11. Healy L., & Hoyles C. (1998). Justifying and proving in school mathematics: Technical report on the nationwide survey. Institute of Education, University of London.
12. Hiebert, J., & Grouws, D. A. (2007). *The effects of classroom mathematics teaching on students' learning.* In F. K. Lester (Ed.), Second handbook of research on mathematics teaching and learning (pp. 371-404). Charlotte, NC: Information Age Publishing.
13. Hsu, H. (2010). *The study of Taiwanese students'experiences with geometric calculation with number (GCN) and their performance on GCN and geometric proof* (doctoral dissertation,)The University of Michigan, Michigan.
14. Howe, K. R.(2001). *Qualitative Educational Research: The Philosophical İssues*. Wahington,DC: American Educational Research Association
15. İskenderoğlu, T. ve Baki, A. (2011). İlköğretim 8.sınıf matematik ders kitabındaki soruların PISA matematik yeterlik düzeylerine göre sınıflandırılması. *Eğitim ve Bilim, 36*(161), 287-301
16. Kilpatrick, J., Swafford, J., & Findell, B. (2001). Adding It Up: *Helping Children Learn Mathematics*. Washington, DC: National Academy Press.
17. Kinach, B. M. (2002). Understanding and learning-to-explain by representing mathematics: Epistemological dilemmas facing teacher educators in the secondary mathematics methods course. *Journal of Mathematics Teacher Education, 5*(2), 153186

18. Kumandaş, H., ve Kutlu, Ö. (2014). Yükseköğretime öğrenci seçmede ve yerleştirmede kullanılan sınavların oluşturduğu risk faktörlerinin okul başarısı üzerindeki etkileri. *Türk Psikoloji Dergisi, 29*(74), 15-31
19. McCrone, S. M. S. & Martin, T. S. (2009). Formal Proof in High School Geometry: Student Perceptions of Structure, Validity, and Purpose. Teaching Proving by Coordinating Aspects of Proofs with Students' Abilities. In Stylianou, D. A.,. Blanton, M. L. & Knuth, E.J. (Eds.), *Teaching and Learning Proof Across Grades: A K-16 Perspective*, (pp. 204-221). New York/Washington, DC: Routledge/National Council of Teachers of Mathematics.
20. Milli Eğitim Bakanlığı [MEB]. (2013). *Ortaöğretim matematik dersi (9, 10, 11 ve 12. sınıflar) öğretim programı.* Ankara: MEB Yayınları.
21. Milli Eğitim Bakanlığı [MEB]. (2018). *Ortaöğretim matematik dersi (9, 10, 11 ve 12. sınıflar) öğretim programı.* Ankara: MEB Yayınları.
22. National Council of Teachers of Mathematics [NCTM]. (2000). *Principles and standards for school mathematics.* Reston, VA: Author
23. Petrou, M. & Goulding, M. (2011). Conceptualising teachers' mathematical knowledge in teaching. In T. Rowland & K. Ruthven (Eds.), *Mathematical Knowledge in Teaching, Mathematics Education Library 50* (pp. 9-25). London: Springer.
24. Pulley, C. A. (2010). *Using instruction to investigate the effects of assessing reasoning tasks on students' understanding of proof* (doctoral dissertation). Illinois State University, Illinois, USA.
25. Shulman, L. S. (1986). Those who understand: Knowledge growth in teaching. *Educational Researcher, 67*,4-14
26. Stylianides, A. J. and Stylianides, G. J. (2009). Proof constructions and evaluations. *Educational Studies in Mathematics, 72,* 237-253. doi: 10.1007/s10649-009-9191-3
27. Tiemann, G. E. (2011). *The impact of a school-wide high school advanced placement program and culture on participating students' high school achievement and engagement outcomes and first year university academic success* (doctoral dissertation). University of Nebraska, USA.
28. Vale, C., McAndrew, A., and Krishnan, S. (2011). Connecting with the horizon: Developing teachers' appreciation of mathematical structure. *Journal of Mathematics Teacher Education, 14*(3), 193-212.
29. Van de Walle, J. A. (2013). *Elementary and middle school mathematics: Teaching developmentally* (7th ed.). Boston: Allyn and Bacon.