INTERNATIONAL JOURNAL OF RESEARCH
In Advanced Engineering Technologies

Volume 6, Issue 3 APR 2018

DESIGN OF AN EFFICIENT FFT-BASED MONTGOMERY MULTIPLICATION

K. Kranthi Kumari[1] **V. Ramesh**[2]
kkranthi409@gmail.com[1] ramesh719@gmail.com
[1]PG Scholar, VLSI, Vaagdevi Institute of technology&science , Proddatur, Kadapa(dist), Andhra Pradesh.

[2] Assistant Professor, Department of ECE, Vaagdevi Institute of technology&science, Proddatur, Kadapa(dist), Andhra Pradesh.

Abstract: Modular multiplication is a basic operation in public key cryptosystems, like RSA and elliptic curve cryptography (ECC). There are many algorithms to speed up its calculation. Among them, Montgomery algorithm is the most efficient method for avoiding expensive divisions. Recently, due to the increasing use of diverse embedded systems, variable precision modular multiplications with scalable architectures gain more and more attentions. In this paper, we propose a new word-based implementation of Montgomery modular multiplication. A predict policy is incorporated with a scalable architecture to reduce area cost and time latency. Compared with other scalable designs, our area-time product is the best among all, with little memory overhead.

Keywords;—MontgomeryModular Multiplication word based version of Montgomery algorithm, scalable architecture.

I. INTRODUCTION

Modular multiplication is regarded as a basic arithmetic operation widely used in many public key cryptosystems, such as RSA and ECC. These cryptographic protocols require the application of large modulus M, which is needed in repeated modular multiplications. Traditional algorithms use division by M to avoid calculation overflow, but it leads to low performance in hardware and software implementations. So far, the most efficient and popular modular multiplication method is Montgomery algorithm. Its modular multiplication starts from the least significant position and uses division by a power of two, instead of division by M. As a result, the primary operations for Montgomery algorithm are just simple additions and shifts. For the sake of high performance, several designs had proposed the use of carry save adder (CSA) to prevent the long carry propagation delay.

To speed up the computation of Montgomery multiplication, various techniques had been reported, such as systolic architecture designs to get high throughput. However, most improved Montgomery modular multiplication methods were developed for fixed precision of operands, which are not applicable for variable precision multiplication.. In Huang selected the result, from two possible values, based on the most significant bit to reduce the clock latency. Besides, a high radix word-based Montgomery algorithm was proposed to decrease the operation time, by using the Booth encoding technique. Nevertheless, its computation logic requirement increases as well.

II. Existing System

A. Montgomery Modular Multiplication

Montgomery modular multiplication algorithm replaces the trial division by simple addition and shift operations. Given an n-bit odd modulus M and a radix constant R, defined as r n mod M. The

radix number is r = 2d , where d is radix parameter. Montgomery modular multiplication (MM) is defined as S = X × Y × R-1 mod M, where X and Y are integers smaller than M. To use Montgomery multiplication in public key cryptosystems, we need to transform X and Y to X × R mod M and Y × R mod M through the use of constant value R2 mod M. The computation S = X × Y × R-1 mod M is repeatedly processed in Montgomery domain. A final step is needed to transform the result back to S = X × Y mod M. The radix-r MM algorithm is described as:

Algorithm 1. Montgomery Modular Multiplication

Inputs: M (odd modulus, n bits), X (multiplier, n bits), Y (multiplicand, n bits), and $X, Y < M$.
Output: $S \equiv X \times Y \times R^{-1} \bmod M$, $R = r^n$, $r = 2^d$, $0 \le S < M$.
1: $S = 0$
2: for $i = 0$ **to** $n - 1$ **do**
3: $S = S + x_i \times Y$
4: $q = (-S \times M^{-1}) \bmod r$
5: $S = (S + q \times M) / r$
6: end for
7: if $(S \ge M)$ **then** $S = S - M$
8: end if
9: return S

Since the convergence range of S is 0 to 2M, the final output subtraction of Montgomery modular multiplication can be directly removed to avoid the extra subtraction S = S – M .

III. Proposed System

FFT-BASED MONTGOMERY MODULAR MULTIPLICATION UNDER MCLAUGHLIN'S FRAMEWORK

In this section, an FFT-based Montgomery modular multiplication under McLaughlin's Framework (FMLM3) proposed, parameter specification of the algorithm is introduced

The Proposed Algorithm of FMLM3.

The FFT method can be applied to Algorithm 1 to perform efficient multiplications modulo R and Q', the FFT-based algorithm (FMLM3) is provided as shown in Algorithm 2. The computation of FMLM3 starts from either time or spectral domain, which depends on the type of input data and the output should be consistent with the input. Algorithm 2 starts the computation from spectral domain, thus two more steps are required to obtain T(t) and T' (t).

In Step 12 of Algorithm 2, the results of NCT^{-1} should be equivalent to the components of NCC, which can be extracted from Equation (4)

Algorithm 2 Proposed FFT based Montgomery Modular Multiplication under Mclaughlin's Frame work(FMLM^3)

Input: Maximum operand size l, satisfy $2^l > N$; R, R^{-1}, N and N' satisfy $RR^{-1} - NN' = 1$, ensure $R = 2^l - 1 > N$, $\gcd(R, N) = 1$, $0 < R^{-1} < N$, $0 < N' < R$; pre-compute $T'(N)$ and $T(N')$; for arbitrary $x, y \in \mathbb{Z}_N$, pre-compute $T(x)$, $T'(x)$, $T(y)$, and $T'(y)$; select $Q' = R + 2 = 2^l + 1$
Output: $T(t), T'(t)$, where $t = xyR^{-1} \pmod N$

Compute $m = xyN' \bmod R$:
1: $T(g) = T(x) \odot T(N') \bmod M$
2: $g = \mathrm{CT}^{-1}(T(g))$
3: $z_0 = g \bmod R$
4: $T(z_0) = \mathrm{CT}(z_0)$
5: $T(z_1) = T(y) \odot T(z_0) \bmod M$
6: $z_1 = \mathrm{CT}^{-1}(T(z_1))$
7: $m = z_1 \bmod R$
8: $T'(m) = \mathrm{NCT}(m)$
Compute $S = xy + mN \bmod Q'$:
9: $T'(z_2) = T'(x) \odot T'(y) \bmod M$
10: $T'(z_3) = T'(m) \odot T'(N) \bmod M$
11: $T'(z_4') = T'(z_2) + T'(z_3)$
12: $z_4' = \mathrm{NCT}^{-1}(T'(z_4'))$
13: Restrict z_4' using Equation (12) to obtain z_4
14: $S = z_4 \bmod Q'$
15: If $2|S$, $s = \frac{2Q'-S}{2} \bmod Q'$, else $s = \frac{Q'-S}{2}$
16: If $xy + mN \equiv s \pmod 2$, $t = s$, else $t = s + Q'$
17: If $t \geqslant N$, $t = t - N$
18: **Return** $T(t) = \mathrm{CT}(t)$
19: **Return** $T'(t) = \mathrm{NCT}(t)$

.

$$z_n = \sum_{i+j=n} x_i y_j - \sum_{i+j=n+P} x_i y_j$$
$$= \sum_{i=0}^{n} x_i y_{n-i} - \sum_{i=n+1}^{P-1} x_i y_{n+P-i},$$

where zn is the nth component of NCC, and i; j; n = 0; 1; ... ; P-1. Clearly, for xi; yj ϵ [0; B), each zn has a lower bound $-(P-1-n)(B-1)^2 \le 0$ and an upper bound $(n + 1)(B-1)^2 > 0$. The lower bound indicates that a negative component may be obtained by NCC. However, all components of z'_4 (Step 12) are within the range of [0; M), since NCT^{-1} is performed in ring ZM. Thus, the

components of z' $_4$ must be restricted to Equation by subtracting M before obtaining z4. This restriction guarantees a correct result of NCC using the FFT method

$$-2(P-1-n)(B-1)^2 \leqslant z_n \leqslant 2(n+1)(B-1)^2.$$

Due to the fact that the sum of two NCCs are computed, where the addition takes place in the Step 11, therefore, both the lower and upper bounds in Equation (12) are doubled.

The FMLM3 is able to use length-P transforms, while the algorithm requires length-2P transforms because of the zero-padding. This is considered to be the major advantage of FMLM3. When computing modular exponentiation x^k mod n using the FMLM3, T' (N) and T(N') can be reused. Besides, the result of xN' mod R (T(z0) in Step 4 of Algorithm 2) can also be reused, this reduces the number of transforms from 7 to 5, more specifically, Steps 1-4 can be saved during the modular exponentiation.

A complexity comparison is provided in Table 1 between the FMLM3 and other modular multiplication methods. RNS refers to the residue number system, which is another divide-and-conquer approach to compute MMM. Since the reduction steps, the complexity of Barrett modular multiplication is estimated by assuming that the multiplication step has digit-level complexity of 4Plog $_2$P + 6P. It can be observed that the FMLM3 has the lowest complexity among all compared algorithms.

Parameter Specifications

In order to perform fast computation of the FMLM3, we choose B = 2^u so that representing an integer in radix-B form is simply a bitwise partition; besides, we select P = 2^v to enable the radix-2 FFT computation. Thus, the maximum supported operand size is defined as l = log $_2B^P$ = u 2^v. By substituting l, R and Q' can be redefined as R = B^P-1 and Q' = B^P +1, respectively. In order to avoid data overflow during the NWT computation, the ring size M must satisfies

$$M > 2(B-1)^2 P.$$

In fact, ensuring M > (B-1) ^{2} P already maintains the precision of modular multiplication. We add one extra bit in (13) so that xy and mN (Step 11 of Algorithm 2) can be component-wisely added prior to the NCT^{-1}. This avoids the long carry chain when performing the addition in time domain. To apply fast reduction computation, modulus M should have low Hamming weight. Thus, we let M to be a Fermat number Fv = 2^P + 1, where P = 2^v.

For a composite M, one can always define a length-2^{v+1-l} NWT with $\omega = 2^{2i}$ and A = 2^{2i-1} over ZM, where i =0; 1; ... ; v + 1. Practically, A should satisfy two conditions:

- A is as small as possible, which results in a larger l;
- A has simple expressions so that multiply by A^k can be computed easily by shifts and additions.

Consequently, the smallest A is $\sqrt{2}$ when i = 0, where A has an expression of

$$A \equiv 2^{2^{v-3}}(2^{2^{v-2}}-1) \bmod 2^{2^v}+1.$$

Table 1 Supported operand size fermat numbers F_n when r=6,7

Max. Operand size l	Ring size M	Transform length P = 2^v	Roots ω / A	Digit bit length u
1,792	$F_6 = 2^{64}+1$	64	4 / 2	28
3,584	$F_6 = 2^{64}+1$	128	2 / $\sqrt{2}$	28
7,680	$F_7 = 2^{128}+1$	128	4 / 2	60
15,104	$F_7 = 2^{128}+1$	256	2 / $\sqrt{2}$	59

Following the previous parameter specifications, we select two Fermat numbers F6 and F7, and summarize the supported parameter sets in Table 2 as examples. The Fermat numbers in Table 2 support the major key sizes of RSA (i.e., 1,024, 2,048, 3,072, 4,096 and 7,680-bit) suggested by the NIST and ECRYPT. However, it is hard to find an appropriate Fv to support an l which is closed or equal to the suggested key size.

In order to narrow the gap between l and the suggested key size, pseudo Fermat numbers F

$= 2^{c2v} + 1$ ($c \geq 2$) are employed to define a length-2^{v+1-i} NWT. Thus, we introduce an extra parameter c, and redefine $M = 2^{cP} + 1$, $\omega = 2^{2c}$ and $A = 2^c$, respectively. Based on (13), c should satisfy

$$c = \frac{1}{2} \geq \frac{v + 2u + 1}{P} \quad \text{or} \quad c = \left\lceil \frac{v + 2u + 1}{P} \right\rceil.$$

The application of pseudo Fermat number and parameter c can provide us more flexible choices when generating the parameter sets. Table 4 provides eligible parameter sets targeting on 2,048-bit key size. Note that the maximum operand size l is exactly 2,048-bit without "wasting" any bit.

OPTIMIZATIONS OF THE FMLM3.

In this section, fast modular reduction algorithms are introduced and a modified version of the FMLM3 is proposed. Based on these optimizations, an efficient parameter set selection method is then summarized.

Modulo R Reduction and Redundant Representation

In Step 2 of Algorithm 2, g is obtained by

$$g = \sum_{i=-u}^{P-1} g_i 2^{ui},$$

where gi is the ith component of CT^{-1}. Since $gi \leq (B-1)^2 P$, the maximum bit length of g is uP + u + v + 1 > l. This implies g may be larger than R, an extra reduction is required to reduce g within the range of [0; R) (Step 3). With $R = 2^{uP}-1$, g mod R can be operated in two steps. First, we compute

$$g' = g_H + g_L = \left\lfloor \frac{g}{2^{uP}} \right\rfloor + (g \bmod 2^{uP}).$$

Since g' equals to either z0 or z0 + R, where $z0 \equiv g \bmod R$, a second step is required to correct the case when g' = z0 + R by subtracting R from g'. It can be observed that providing z0 + R in Step 3 is tolerable, since the remaining extra R can be reduced by the second modulo R reduction in Step 7. Therefore, Step 3 of Algorithm 2 can be computed by only one addition as shown in Equation (17), and the correction step can be saved.

To perform fast computation of Equation (17), we introduce the redundant representation. For radix-B representation of x, each xi has u bits. If xi maintains one extra bit precision (u + 1 bits) on the same basis, then we call x is in its redundant representation. There are more than one radix-B redundant representation for each x. Taking x = 871; 206 and $B = 2^5$ as an example, the radix-B representation of x is (x0; x1; x2; x3)=(6; 25; 18; 26), while the radix-B redundant representation of x can be either (38; 24; 50; 25) or (6; 57; 49; 25).

When applying redundant representation to compute Equation (17), we add the two operands digit-by-digit, and prevent the carry bit from propagating to the next digit level addition. Thus, the sum of each two digits has maximumly u + 1 bits, the g' is then in its redundant form. Under such circumstances, M must satisfy $M > P(B-1)(2B-2) = 2P(B-1)^2$, which is same as the condition in (13). Therefore, no additional restriction of M is required when applying redundant representation.

Table 2 Examples of eligible parameter sets targeting *l*=2048

Set no.	$B = 2^u$	$P = 2^v$	c	M	w / A
1.	2^{128}	16	17	$2^{272}+1$	$2^{34} / 2^{17}$
2.	2^{64}	32	5	$2^{160}+1$	$2^{10} / 2^5$
3.	2^{32}	64	2	$2^{128}+1$	$2^4 / 2^2$
4.	2^{16}	128	0.5	$2^{64}+1$	$2 / 2^{48} - 2^{16}$
5.	2^8	256	0.5	$2^{128}+1$	$2 / 2^{96} - 2^{32}$
6.	2^4	512	0.5	$2^{256}+1$	$2 / 2^{192} - 2^{64}$

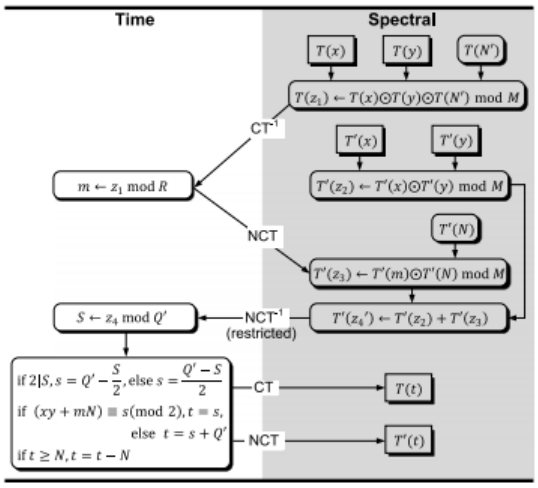


Fig.1 Data flow of the modified FMLM3 (Algorithm 3), the right-angled blocks represent input/output data, T(N') and T0 (N) are pre-computed.

Algorithm 3 Modified Version of proposed FMLM[3]

Input: Requirement is the same as Algorithm 2
Output: $T(t), T'(t)$, where $t = xyR^{-1} \pmod N$
Compute $m = xyN' \bmod R$:
1: $T(z_0) = T(x) \odot T(N') \bmod M$
2: $T(z_1) = T(y) \odot T(z_0) \bmod M$
3: $z_1 = \mathrm{CT}^{-1}(T(z_1))$
4: $m = z_1 \bmod R$
5: Step 8-19 of Algorithm 2

This acceleration approach is only available for Step 3 of Algorithm 2. The second modulo R reduction in Step 7 is followed by a different modulus Q', so an accurate result of the reduction is required.

Modulo M Reduction

Modulo M reduction is one of the basic operations in the FMLM3. As introduced in [29], modulo M reduction requires two steps: first, divide operand x into digits on radix-cP basis and computed

$$x \bmod M = \sum_{i=0} (-1)^i x_i.$$

Reduce the Number of Transforms in the FMLM3 Algorithm

In Algorithm 2, the computation of T' (m) requires 4 NWTs, since T(x), T(y) and T(N') are multiplied sequentially. This number can be reduced to 2 by multiplying the three operands "all at once", as shown in Algorithm 3. Fig. 4.1 provides the data flow of the modified FMLM3.

Compared with Algorithm 2, the modified version reduces the number of NWTs from 7 to 5. Benefit from this improvement, the Algorithm 3 has a lower complexity and a simpler data flow. As a trade-off, a larger dynamic range of M is required

$$M > P^2(B-1)^3,$$

accordingly

$$c = \frac{1}{2} \geqslant \frac{2v+3u}{P} \quad \text{or} \quad c = \left\lceil \frac{2v+3u}{P} \right\rceil.$$

Compared with (13), for the same values of u and v, a larger M may be obtained in order to satisfy new restriction (19). Taking the parameter sets in Table 4 as examples: when applying Set 1 to Algorithm 3, c and M will be enlarged to 25 and 2^{400} + 1, respectively. However, Sets 3-6 can be applied directly to Algorithm 3 with no changes. This implies a "free" number reduction of NWTs from 7 to 5.

Efficient Parameter Set Selection

The efficiency of FMLM3 is impacted by the parameters, especially by the transform length P and ring size M. A larger P implies more digit-level multiplications; a larger M implies larger operands of each digit-level operation. Fig. 4.2 is depicted in order to explore the relation between the parameters in Table 4 for l = 2; 048.

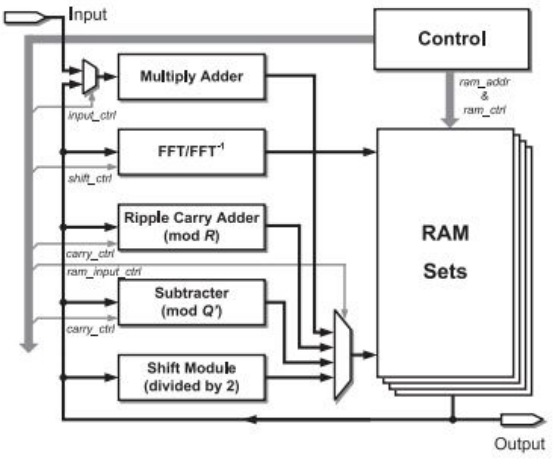


Fig.2. Top level architecture of the proposed FMLM3.

The Control unit generates one control code at each clock cycle, control code includes all the necessary control signals, for instance: ram ctrl

signals control the behavior of each RAM; shift ctrl signals control bit-wise shifting during the transforms.

In Fig. 4.2, the value of P increases along with the increase of v. While M decreases until c = 0:5. However, after c = 0:5, both M and P go up with the increase of v, thus, the complexities of the corresponding parameter sets are always higher than the previous ones. For example, when v = 6 (Set 3 in Table 4) and v = 8 (Set 5), a same M is required, but the transform length of Set 3 (P = 64) is shorter than that of Set 5 (P = 256). Clearly, Set 3 requires less digit-level multiplications, and therefore, Set 3 is more efficient than Set 5. In summary, the parameter sets in the shadowed area in Fig. 4.2 are all considered to be inefficient. However, hardware realizations are required to further evaluate the efficiency of rest parameter sets. We generalize the analysis above and proposed a parameter set selection method in Algorithm 4 to avoid the inefficient sets as just discussed.

Algorithm 4 Efficient parameter set selection method

Input: Targeted operand size l
Output: Parameter sets $\{l, v, u, P, B, c, M, w, A\}$
1: Let $P = 2^v$, v is positive integer, traversal eligible P from 2 to l, when $P = 1$, the FMLM[3] is equivalent to schoolbook multiplication.
2: Calculate digit size $u = \lceil \frac{l}{P} \rceil$ and radix $B = 2^u$.
3: Generate $R = B^P - 1$ and $Q = B^P - 1$.
4: If $\frac{v+2u+1}{P} \leqslant \frac{1}{2}$, $c = \frac{1}{2}$, else $c = \lceil \frac{v+2u+1}{P} \rceil$; if $\lceil \frac{2v+3u}{P} \rceil = \lceil \frac{v+2u+1}{P} \rceil$, apply Algorithm 3, else apply Algorithm 2.
5: Let $M = 2^{cP} + 1$, $w = 2^{2c}$ and $A = 2^c$ are primitive Pth root of unity and (-1), respectively.
6: Update $l = u \cdot 2^v$, l is the practical maximum operand size, record $\{l, v, u, P, B, c, M, w, A\}$ as an eligible parameter set.
7: If $c = \frac{1}{2}$, stop selection, else repeat 1-6.

FFT/FFT1 Unit.

The architecture of our design is targeted on high clock frequency while maintaining a small resource cost. The pipelined butterfly structure (BFS) is used in our FFT/FFT^{-1} unit to achieve this goal. Instead of the in place FFT, the constant geometry FFT is applied to the FFT computation. Compared with the in-place FFT, constant geometry FFT has a same connection network between every adjacent stages, which results in a simpler read-and-write control. In order to explored the trade-off between hardware resources and latency, both architectures with 1 and 2 BFSs are built. Fig. 4.4 provides the pipelined architecture of the FFT/FFT1 unit which integrates 2 BFSs.

Since the FMLM3 employs two types of NWTs, CT and NCT, a more complex architecture is designed. Apart from the two BFSs, a Channel Switcher (CS), which consists of eight 2-to-1 MUX arrays is designed to ensure the intermediate digits can be written into the correct RAM location. Moreover, two Final Stage Operators (FSO A and B) are designed to compute the final stage of the inverse NWTs. Note that during the final stage computation, the results of channel 0 and 1 are forwarded to the FSOs first, concurrently, the results of channel 2 and 3 are stored into the buffer and will be operated after completing the computations of channel 0 and 1. The FFT/FFT^{-1} unit is designed with six inputs, four inputs forward the digits into BFSs for the FFT computation, the other two inputs forward the pre-computed upper bound constraints into FSOs to restrict the results of NCT^{-1} before entering the final accumulation. Before applying the constraints (12) of NCT1, the unconstrained digits are non-negative and equal to either xn (constraint met) or xn + M (constraint not met). Since the lower bounds of (12) are nonpositive integers, while xn ∈ [0;M), we only need to check the upper bounds and correct the xn + M cases by subtracting M.

An accumulator is designed to recombine the results of CT^{-1} or NCT^{-1}. Two adjacent digits are added at each cycle and the results are generated in two pipeline stages. In the first stage, it computes

$$R_0 = x_{i+1}2^u + x_i,$$

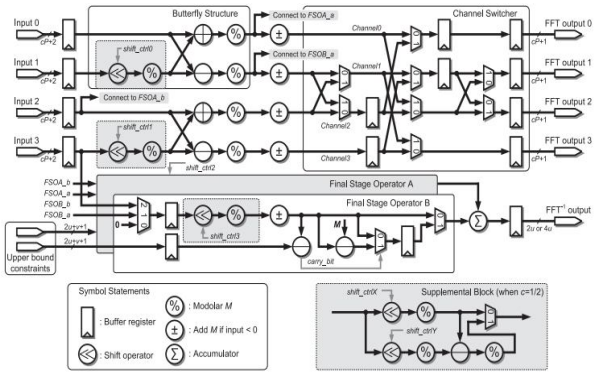


Fig.3 Pipelined architecture of FFT/FFT^{-1} with two butterfly structures.

shift ctrl# signals are responsible for the multiplication of powers of 2 during the transform. The three dashed blocks are replaced by the Supplemental Block when c = 1/2. A Channel Switcher is used to reorder the output digits, and ensure operations of the following stage can be computed correctly. Switch control signals of the MUXs and registers following the operators are omitted. The Final Stage Operators A and B are responsible for the last stage of CT^{-1} and NCT^{-1}.

where xi and xiþ1 denotes the two input digits. In the second stage, R0 is added to the accumulation register and two radix-B digits are generated

$$r_{i+1} = R_0 + \left\lfloor \frac{r_i}{2^{2u}} \right\rfloor,$$
$$X_{i+1}2^u + X_i = r_i \bmod 2^{2u},$$

where Xi, Xi+1 denote the two output digits, ri denotes the data stored in accumulation register (r0 = 0). It worth to note that in our design, the multiplication, division and reduction in both (21) and (22) are performed efficiently by shift or bitwise partition.

Shift operators are designed to compute the times powerof-2 operations during the NWT computation. Control signals shift ctrl0 and shift ctrl1 transmit the twiddle factors (the number of ω) to handle the shift operation. The jth stage shift bits are obtained according to Equations (7), (8), and (9). Since the inverse NWTs are scaled by P^{-1} or (A^{n}P)$^{-1}$ (when n = 0, (A^{n}P)$^{-1}$ = P^{-1}), one more shift operator is integrated in each of the FSOs (controlled by shift ctrl2 and shift ctrl3, respectively). When c≠ 1/2, A=2^c is a rational number, and ω has integer powers in Equations (7), (8), and (9). NWTs can be computed by the FFT/ FFT^{-1} unit. When c = 1/2, A ≡ $\sqrt{2}$ ≡ $2^{3\text{-}2^{v-3}}$ - $2^{2^{v-3}}$ mod M, therefore, one subtraction, two shifts and three modulo M reductions are required to multiply A.

For the case c = 1/2, the computation of NCT and NCT^{-1} require more operations compared to CT and CT^{-1} due to the non-integer power of ω and the scale of the irrational number. Considering the NCT computation when A = $\sqrt{2}$, the twiddle factors are obtained by $\omega^{-\left[\frac{n}{J}\right]\cdot J+J/2}$ according to Equation (8), where J = 2^{v-1-j} . This indicates the power of ω is not an integer only in the final NCT stage (j = v-1). The MUX in the Supplemental Block will select the lower output during the final stage computation while selecting the upper one for the rest of the stages. By substituting J = 1 and ω = 2, shift bits of the final stage is obtained by

$$\omega^{-\lfloor n/J \rfloor \cdot J + J/2} = 2^{3\cdot 2^{v-3}-n} - 2^{2^{v-3}-n} \bmod M.$$

Considering the NCT^{-1} computation when A = $\sqrt{2}$, since NCT^{-1} is scaled by (A^{n}P)$^{-1}$, the final stage of NCT^{-1} is divided into two cases. When n is even, A^n = 2 $^{n/2}$ is always an integer, so (A^{n}P)$^{-1}$ ≡ $2^{2v\text{-}v\text{-}n/2}$ mod M is simply a power of 2. However, when n is odd, computing (A^{n}P)$^{-1}$ is more complicated

$$(A^n P)^{-1} \equiv (2^{3\cdot 2^{v-3}-1} - 2^{2^{v-3}-1}) \cdot 2^{2^v - v - \frac{n-1}{2}} \bmod M.$$

Therefore, when A =$\sqrt{2}$, FSO A remains no change since n is always even in this channel, while the dashed part of FSO B will be replaced by the Supplemental Block.

Extension.

In the extension Pasta adder is used in the place of Ripple carry Adder. Since Pasta adder is the fastest adder than the ripple carry adder speed is increased.

Design of PASTA.

An adder or a summer is a digital circuit that performs addition of numbers. Addition forms the basis for many processing operations, from counting, multiplications to filtering. In processors adders are also used to increment program counters, calculate effective addresses, table indices and similar operations. Thus the performance of processor is greatly influenced by the speed of the adders used by the processor

The basic building block of combinational digital adders is a single bit adder. The simplest single bit adder is a half adder (HA). The full adders (FA) are single bit adders with the carry input and output. The full adders are basically made of two half adders in terms of area, interconnection and time complexity.

Apart from the theoretically possible best design for adders, some implementation problems regarding circuit complexity and fabrication limitations also play a vital role in circuit design. The circuit complexity and irregular design can render it infeasible for VLSI fabrication.

In this section, the architecture and theory behind PASTA is presented. The adder first accepts two input operands to perform half additions for each bit.

A. Architecture of PASTA

The general architecture of the adder is shown in Fig. 4.6. The selection input for two-input multiplexers corresponds to the Req handshake signal and will be a single 0 to 1 transition denoted by SEL. It will initially select the actual operands during SEL=0and will switch to feedback/carry paths for subsequent iterations using SEL=1. The feedback path from the HAs enables the multiple iterations to continue until the completion when all carry signals will assume zero values

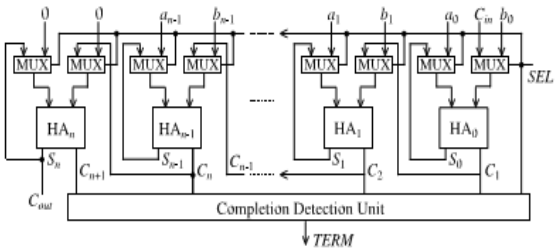


Fig. 4.General block diagram of PASTA

B. State Diagrams

In Fig. 4.7, two state diagrams are drawn for the initial phase and the iterative phase of the proposed architecture. Each state is represented by (Ci+1Si) pairwhereCi+1, Si represent carry out and sum values, respectively, from the ith bit adder block. During the initial phase, the circuit merely works as a combinational HA operating in fundamental mode. It is apparent that due to the use of HAs instead of FAs, state (11) cannot appear.

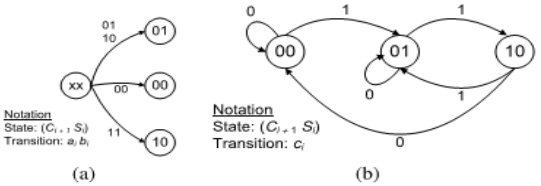


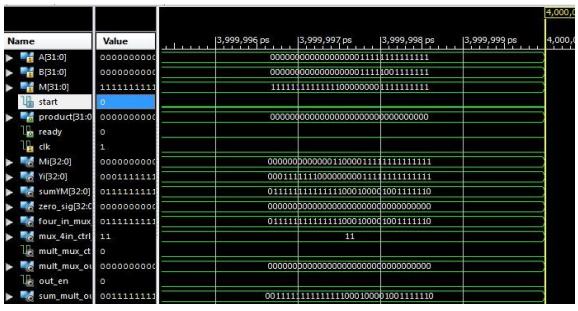
Fig.5. State diagrams for PASTA. (a) Initial phase. (b) Iterative phase

IV. Simulation Results

Results of the project are observed using XILINX software.

Proposed (CSA).

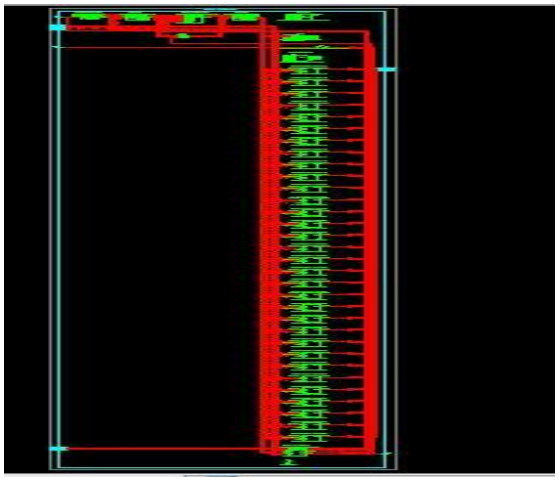
MMM Simulation for CSA adder:



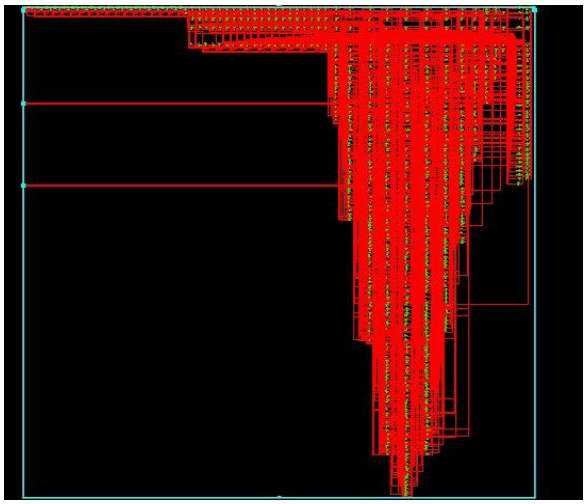
Synthesis Results:

In synthesis Implementation of the project is done where we get RTL and Technology Schematic.

RTL Schematic:



Technology Schematic:



DESIGN SUMMARY:

Here area of the project is shown in different sections as table.

Logic Utilization	Used	Available	Utilization
Number of slices	**286**	**4656**	**6%**
Number of slice Flip-flops	**75**	**9312**	**0%**
Number of 4 input LUTS	**539**	**9312**	**5%**
Number of bonded iobs	**131**	**232**	**56%**

Conclusion:

We proposed a modified version of the FFTbased Montgomery modular multiplication algorithm under McLaughlin's framework (FMLM3). By applying cyclic and nega-cyclic convolutions to compute the modular multiplication steps, the zero-padding operation is avoided and the transform length is reduced by half compared to the regular FFT-based multiplication. Furthermore, we explored for some special cases, the number of transforms can be further reduced from 7 to 5 without extra computational efforts, so that the FMLM3 can be further accelerated.

Future Scope:

Alternative Approach of improving the process of multiplication to reduce the number of steps which improves delay of the circuit and also reduces the complexity is to change the adders using in the circuit. We can also change the length twiddlefactors in Butterflystructures. Furthermore, we explored for some special cases, the number of transforms can be further reduced from 7 to 5 without extra computational efforts, so that the FMLM3 can be further accelerated.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems," Commun. ACM, vol. 21, no. 2, pp. 120–126, 1978.

[2] R. L. Rivest, "A description of a single-chip implementation of the RSA cipher," Lambda, vol. 1, no. Oct.–Dec., pp. 14–18, 1980.

[3] "Recommendation for key management," NIST, Tech. Rep. Special Publication 800-57, Part-1, Rev.-3, 2012.

[4] P. L. Montgomery, "Modular multiplication without trial division," Mathematics Comput., vol. 44, no. 170, pp. 519–521, 1985.

[5] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," Soviet Physics Doklady, vol. 7, 1963, Art. no. 595.

[6] S. A. Cook and S. O. Aanderaa, "On the minimum computation time of functions," Trans. Amer. Math. Soc., vol. 142, pp. 291–314, 1969.

[7] A. Schonhageand V. Strassen, "Schnelle multiplikation Großer € Zahlen," Computing, vol. 7, no. 3/4, pp. 281–292, 1971.

[8] M. Furer, "Faster integer multiplication," € SIAM J. Comput., vol. 39, no. 3, pp. 979–1005, 2009.

[9] D. Harvey, J. van der Hoeven, and G. Lecerf, "Even faster integer multiplication," CoRR, vol. abs/1407.3360, 2014. [Online]. Available: http://arxiv.org/abs/1407.3360

[10] S. Covanov and E. Thome, "Fast arithmetic for faster integer multiplication," CoRR, vol. abs/1502.02800, 2015. [Online]. Available: http://arxiv.org/abs/1502.02800

[11] A. F. Tenca and C¸ . K. Koc¸, "A scalable architecture for modular multiplication based on Montgomery's algorithm," IEEE Trans. Comput., vol. 52, no. 9, pp. 1215–1221, Sep. 2003.

[12] M. D. Shieh and W. C. Lin, "Word-based Montgomery modular multiplication algorithm for low-latency scalable architectures," IEEE Trans. Comput., vol. 59, no. 8, pp. 1145– 1151, Aug. 2010.

[13] M. Morales-Sandoval and A. Diaz-Perez, "Scalable GF(p) montgomery multiplier based on a digit–digit computation approach," IET Comput. Digit. Techn., vol. 10, no. 3, pp. 102–109, May 2015.

[14] M. Huang, K. Gaj, and T. El-Ghazawi, "New hardware architectures for Montgomery modular multiplication algorithm," IEEE Trans. Comput., vol. 60, no. 7, pp. 923–936, Jul. 2011.