International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

An Efficient Architecture for Interleaved Modular Multiplication

Ahmad M.AbdelFattah[*] *Ayman M.Bahaa El-Din*[*] *Hossam M.A.Fahmy*[*]

***Abstract*—Modular multiplication is the basic operation in most public key cryptosystems, such as RSA, DSA, ECC, and DH key exchange. Unfortunately, very large operands (in order of 1024 or 2048 bits) must be used to provide sufficient security strength. The use of such big numbers dramatically slows down the whole cipher system, especially when running on embedded processors.**

So far, customized hardware accelerators - developed on FPGAs or ASICs - were the best choice for accelerating modular multiplication in embedded environments. On the other hand, many algorithms have been developed to speed up such operations. Examples are the Montgomery modular multiplication and the interleaved modular multiplication algorithms. Combining both customized hardware with an efficient algorithm is expected to provide a much faster cipher system.

This paper introduces an enhanced architecture for computing the modular multiplication of two large numbers X and Y modulo a given modulus M. The proposed design is compared with three previous architectures depending on carry save adders and look up tables. Look up tables should be loaded with a set of pre-computed values. Our proposed architecture uses the same carry save addition, but replaces both look up tables and pre-computations with an enhanced version of sign detection techniques. The proposed architecture supports higher frequencies than other architectures. It also has a better overall absolute time for a single operation.

***Keywords*—Montgomery multiplication, modular multiplication , efficient architecture, FPGA, RSA**

I. Introduction

One common drawback of all public key cryptographic algorithms - though being highly secure - is the heavy computation involved in key generation, digital signature, and data encryption/decryption schemes. Such complexity refers to the use of modular exponentiation in most of the above schemes, taking into account that operands are not less than 1024 bits long (except for Elliptic Curve Cryptography(ECC) which uses modular multiplication of 521 bits numbers as maximum).

Modular multiplication is the heart of modular exponentiation. Accelerating modular multiplication will raise the efficiency of the whole public key cryptosystem. General purpose processors consume thousands of cycles to finish a single operation using classical methods. Many algorithms have been developed to efficiently perform the computation ($X \times Y mod M$) without doing the ordinary pencil-and-paper steps. Examples are Montgomery modular multiplication [1] and interleaved modular multiplication [2]. Now considering embedded environments, such as FPGAs or ASICs, we can achieve higher efficiency by selecting one of the above algorithms to run over special purpose hardware architecture.

In this work, we introduce an enhanced architecture for an interleaved modular multiplier. The proposed design scores better timing than other architectures presented in [3] and [4]. Most previously introduced architectures depend on the use of look-up tables, to provide some kind of caching the operands. Using look-up tables slows down the maximum frequency the hardware can support. Our proposal replaces caching with an improved sign detection technique [5]. Compared to architectures presented in [3], higher frequencies were supported on the same FPGA. The overall operation time is also improved.

Sections II and III introduce the algorithms of modular exponentiation and interleaved modular multiplication. Section IV presents an architecture of an interleaved modular multiplier presented in [3]. Section V introduces a modified version of the sign estimation technique presented in [5]. It also proves the correctness of the new technique. Section VI proposes the modified interleaved modular multiplier based on the new sign estimation technique. It also summarizes the implementation results in comparison with the architecture presented in [3]. The document ends with a conclusion in section VII.

II. Modular Exponentiation

Modular exponentiation means to compute R where,

$$R = X^e \bmod M$$

If we consider $X, e, \& M$ to be in order of 1024bits or higher, the operation becomes very time and space consuming for the ordinary pencil-and-paper method. Therefore, we use the famous square and multiply algorithm shown in Table I. The main operation included in the algorithm is the modular multiplication (lines 3, 4).

The square and multiply algorithm scans the exponent bits, starting by the MSB. For a '1' bit, two modular multiplication are done: squaring the previous result (line 3), and a modular multiplication by X (line 4). For a '0' bit, only squaring is performed. Therefore, Modular exponentiation uses 2048(worst case) or 1.5 $\times$1024 = 1536(average) modular multiplications. This proves the importance of the need to optimize both time and space of modular multiplication.

In the next section, we introduce the interleaved modular multiplication algorithm which will be used in the square and multiply algorithm.

[*] Department of Computer and SystemsEngineering,ASU,Egypt.
[*] ahmad.abdel-fattah@eng.asu.edu.eg
[*] ayman.bahaa@eng.asu.edu.eg
[*] hossam.fahmy@ieee.org

International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:3, No:8, 2009

TABLE I
SQUARE AND MULTIPLY ALGORITHM

Input: X, e, M ; *n-bit numbers* ; $0 \leq X, Y \leq M$ ***Output:*** **:** $R = X^e modM$
n: number of bits of e e_i :i^{th} bit of e

1. $R = 1$
2. for ($i = n - 1$; $i \geq 0$; $i = i - 1$)
{
3. $R = R \times R$ mod M
4. if (e_i=1) Then $R = R \times X$ mod M
}

III. Interleaved Modular Multiplication

Classical modular multiplication means to compute the product ($X \times Y$), and then reduce the result via division by the given modulus (M). Interleaved modular multiplication allows multiplication and reduction to overlap. The intermediate results produced after every iteration are first reduced to the range [0, M-1] before resuming the multiplication process. The original algorithm is shown table II:

TABLE II
INTERLEAVED MODULAR MULTIPLICATION ALGORITHM

Input: X, Y, M ; *n-bit numbers* ; $0 \leq X, Y \leq M$ ***Output:*** $P = X \times Y modM$
n : number of bits of Y x_i : i^{th} bit of X

1. $P = 0$
2. for ($i = n - 1$; $i \geq 0$; $i = i - 1$)
{
3. $P = 2 \times P$
4. $I = x_i \times Y$
5. $P = P + I$
6. if ($P \geq M$) then $P = P - M$
7. if ($P \geq M$) then $P = P - M$
}

The algorithm scans X starting from the MSB. At each iteration, a single Y is conditionally added to the accumulator P according to the bit x_i. At the end of every iteration, P must be guaranteed to be less than M. The condition ($P \geq M$) has to be checked at most twice. This is because P is incremented twice per iteration at worst case, once by P (line 3) and another conditional increment by Y (lines 4 & 5).

The main operations found in the algorithm are:
1. One bit left shift (Line 3).
2. Addition (Line 5).
3. Comparison (Line 6, 7).

Operation (1) is simple and needs no optimization. Operation (2) involves adding two large numbers (1024-2048 bits). Ripple carry adders have the defect of the carry propagation through the whole number to produce the result. Using ripple carry adders will slow down the system maximum operating frequency. Carry look-ahead adders can be used instead. But they have the defect of extensive hardware used to avoid carry propagation. Carry save adders (CSAs) are also a better choice for addition. CSAs occupy much less hardware than carry look-ahead adders but they provide the result in a (Sum, Carry) pair representation. So far CSAs were selected in all improvements of modular multipliers. Operation (3) needs a comparator between P and M. Worst case comparison will also results in a big propagation delay, as we will have to compare all bits from both numbers. The comparison needs both numbers to be in their final form, which conflicts with the CSAs output form.

In the next section, we introduce the latest proposed architecture to optimize the interleaved modular multiplication. The related work tried to compromise between using CSAs and avoiding ordinary comparison. More details can be found in [3] and [6].

IV. Related Work

Figure 1 shows a previously proposed architecture in [3] for the interleaved modular multiplication algorithm shown in Table II. The multiplier uses a single CSA and a look-up table.

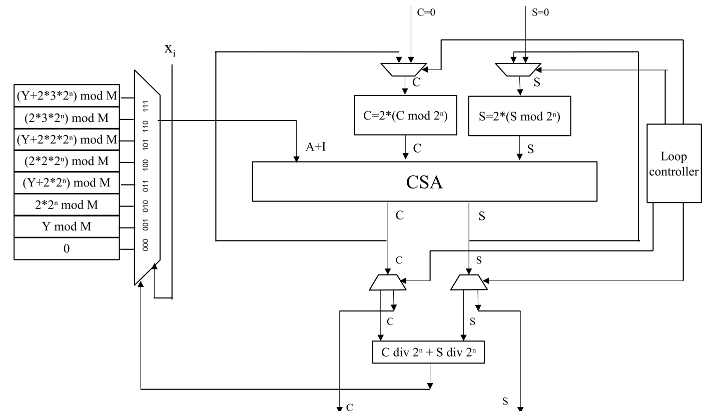

Fig. 1. Redundant Interleaved Modular Multiplier

Referring to Table II, the above architecture solved the comparison problem by a comparison with 2^n instead of M. Another problem is the number of additions or subtractions in the steps (5), (6), and (7) of the algorithm. All these operations were replaced by one addition as follows: Pre-estimation of the number of Ms to be subtracted is done. Another decision to be made is addiny Y or not according to the next iteration of the loop. All possible values for this estimation are pre-computed and stored in the look-up table. In each iteration of the loop, the estimation of the previous iteration can be added to the intermediate result. This architecture achieved 69MHz maximum frequency for 1024 bit operands on a Xilinx Virtex 2 FPGA. A single modular multiplication takes only 14.7 μs [6] +

the time for pre-computations to take place.

As shown in Figure 1, pre-estimated values depend on both $YandM$. Now referring to Table I, square and multiply algorithm requires pre-estimation before every multiplication. The time overhead due to loading the look-up table becomes significant with respect to the hardware operation time.

In our proposed architecture, we also make use of CSAs. CSAs have a drawback of not producing the result in its final form. We introduce a new technique to perform the comparison steps on the (Sum, Carry) pair representation. Comparison will be replaced by evaluating the sign of the quantity $(P - M)$. The Sign determination will be done using the 'Sign Detection' technique explained in the next section.

V. Sign Detection Technique

A. Mathematical Representation

Sign estimation means to decide the sign of a binary number represented in a (Sum, Carry) pair without finalizing the addition. The decision is based on a segment of the sum and a segment of the carry of the same bit length. The longer the segment, the higher the probability of estimating an exact sign will be. It should be noted that both sum and carry are in the two's complement representation.

Previous work [5] proposed a technique for the sign estimation. The technique produces three possible outputs: positive (+), negative (-), and unsure($\pm$). The unsure state is reached when the number is either too large or too small. The usage of such technique in modular reduction was limited to a segment of only 2 bits length[5]. Using such short segments may result in frequently reaching the unsure state.

In our contribution, we have enhanced the sign estimation module to produce a 'Sign Detection' technique, that is, an exact sign will be produced for any number. The proposed module can avoid the unsure state. It can be used with any segment length in modular multipliers. If the segment is of enough length (typically 8-16 bits for 1024bit numbers), the sign will be produced in a single clock cycle. Otherwise, several cycles can be consumed before producing the exact sign. The module works as follows:

A window (segment) of length (w) will be taken from both sum and carry. The window count starts from the MSB and moving downwards. Using this scheme, any n-bit number (X) can be represented as the sum of two quantities:

1. An always-positive number containing bits (0) to $(n-w-1)$ of X. "Note that $(n > w)$"
2. A number in the two's complement representation having bits (0) through $(n-w-1)$ reset to zeros and bits $(n-1)$ through $(n-w)$ as the same as X.

So we can write,

$$X = W(x) + R(x)$$

Table III shows the representation of X for sign detection.

TABLE III

Sign detection representation

$X =$	x_{n-1}	x_{n-2}	$\cdots$	x_{n-w}	$\cdots$	x_1	x_0
$W(x) =$	x_{n-1}	x_{n-2}	$\cdots$	x_{n-w}	0	$\cdots$	0
$R(x) =$	0	0	$\cdots$	0	x_{n-w-1}	$\cdots$	x_0

Note that $R(x)$ is always positive. $W(x)$ holds the sign of the original number. Bits $(n-1)$ through $(n-w)$ of both sum and carry are fed to the sign detection module. The 'detect sign' function DS(S, C) can now be defined.

If X is an n-bit number represented in (S, C) pair:

1. Let $W(s)$, $W(c)$ be the windows taken from S, C respectively.
2. Let $Temp = W(s) + W(c)$.
3. If the MSB of $Temp$ is '0' then X is positive.
4. Else if $Temp \neq$ "$11 \cdots 11$" then X is negative.
5. Else($Temp$ is all ones), request another carry save addition. Update (S, C). Go to (1).

Step 5 avoids the unsure state by feeding (S, C) pair back to the CSA, thus requesting another (S, C) representation of the same number. This is an interesting feature of CSAs that we can produce many (Sum, Carry) pairs all referring to the same result. The function can now restart to produce an exact sign.

In the next subsection we prove the correctness of the new sign detection technique.

B. Correctness of Sign Detection Technique

In this subsection, we prove the correctness of the sign detection technique. Proof will be introduced for window lengths of 2, 3, and 4 bits. It will be shown that it can be generalized for any number of bits.

Proof: It's clear that

$$R(s) \geq 0, R(c) \geq 0$$

Since

$$X = W(s) + W(c) + R(s) + R(c)$$

Then if

$$Temp = W(s) + W(c) > 0$$

which means that MSB of Temp is zero, then X is positive.

For the case MSB of Temp is one (Temp < 0), we will tabulate the possible negative values of Temp added to the maximum possible values of the sum $(R(s) + R(c))$. Since both $R(s)$ and $R(c)$ are of length $(n - w)$, then the maximum value of their sum is equal to $2 \times (2^{n-w} - 1)$. We will consider three cases for window lengths of 2, 3, and 4 bits as shown in tables IV, V, and VI.

We note that unless the least w bits (Where w is the window length) of $Temp$ is all ones, then X will be negative regardless of the summation $R(s) + R(c)$. Otherwise, we need the exact summation $(R(s) + R(c))$ to determine the sign. To avoid the summation of such long operands, we will perform a single carry save addition to obtain a new

TABLE IV
Temp is negative, Window Length = 2 bits

$W(s)+W(c)$ "least 2 bits"	$Temp$	$X = Temp + 2 \times 2^{n-w} - 2$
10	$-2 \times 2^{n-w}$	negative
11	$-1 \times 2^{n-w}$	positive

TABLE V
Temp is negative, Window Length = 3 bits

$W(s)+W(c)$ "least 3 bits"	$Temp$	$X = Temp + 2 \times 2^{n-w} - 2$
100	$-4 \times 2^{n-w}$	negative
101	$-3 \times 2^{n-w}$	negative
110	$-2 \times 2^{n-w}$	negative
111	$-1 \times 2^{n-w}$	positive

(Sum, Carry) pair of X and restart the sign detection. So the technique is proved to produce an exact sign. ■

The following notes should be taken into account:

- If the window is large enough, then we may not need to restart the function at all. Typical values of the window lengths are 8-16 bits for 1024 bit numbers.
- A ripple carry adder of w-bit length is needed to perform the exact summation of W(s)+W(c). This will not affect the speed of the hardware since w is sufficient to be 8 or 16 bits.
- It's not efficient to detect the sign of an (S, C) pair placed in registers of larger size. For example, detecting the sign of 16-bit (S, C) pair is not efficient if we are using 32 bit registers or higher. This will cause the sign detection to restart several times.

The next section introduces the modified interleaved modular multiplier. It also shows how the sign detection technique will be used for modular multiplication.

VI. Modified Interleaved Modular Multiplication

A. Algorithm and Architecture

The algorithm shown in table VII combines both carry save addition and sign detection to enhance the performance of the interleaved modular multiplication.

The modified interleaved modular multiplication differs from the original algorithm as follows:

- Addition and subtraction are performed using CSAs.
- Comparisons made in steps (6, 7) are performed by detecting the sign of the quantity $(P - M)$. If the DS function returns (+), then subtraction is approved. Otherwise, subtraction must be denied to retrieve the value of P. This can be easily done by saving P before doing the subtraction. Figure 2 shows the architecture of the new multiplier.

B. Implementation and Results

The main sub-modules found in the interleaved modular multiplier are:

TABLE VI
Temp is negative, Window Length = 4 bits

$W(s)+W(c)$ "least 4 bits"	$Temp$	$X = Temp + 2 \times 2^{n-w} - 2$
1000	$-8 \times 2^{n-w}$	negative
1001	$-7 \times 2^{n-w}$	negative
1010	$-6 \times 2^{n-w}$	negative
1011	$-5 \times 2^{n-w}$	negative
1100	$-4 \times 2^{n-w}$	negative
1101	$-3 \times 2^{n-w}$	negative
1110	$-2 \times 2^{n-w}$	negative
1111	$-1 \times 2^{n-w}$	positive

TABLE VII
Modified Interleaved Modular Multiplication

Input: X, Y, M ; n-bit numbers ; $0 \leq X, Y \leq M$
Output: $P = X \times Y mod M$

n: number of bits of X
x_i : i^{th} bit of X

1. $S = 0$; $C = 0$;
2. for (i = n - 1; i $\geq$ 0; i = i - 1)
{
3. $S = 2 \times S; C = 2 \times C$;
4. $I = x_i \times Y$
5. $(S, C) = S + C + I$
6. if $DS(P - M)$ = +ve Then $P = P - M$
7. if $DS(P - M)$ = +ve Then $P = P - M$
}

1. Carry Save Adder
2. Sign Detection Module
3. The Loop Controller

All sub-modules and system top level entity were described using VHDL. Behavioral simulation was done to verify the functionality of the description. Verification was accomplished according to a set of test cases developed by a software package for big number calculations. Synthesis, place and route, and post route simulation were done on the Xilinx XC2V3000-6 FPGA. Tables VIII and IX compare between our proposed modular multiplier and the redundant multiplier introduced in [3]. The redundant interleaved multiplier was proved to be better than two version of the Montgomery multiplier[3]. Although our proposed architecture occupies more configurable logic block (CLB) slices, an impressive improvement in the maximum operating frequency was achieved. This is due to the replacement of the look up table with the sign detection module. The tables summarize both device utilization (percentage slices out of 19200) and maximum operating frequency for bit lengths 32, 64, 128, 256, 512, and 1024 bits. The window used for the sign detection was fixed at 16 bits.

Further improvement was accomplished considering the absolute time per operation. One drawback of the architecture of [3] is the need to load the look up table each time before multiplication. Referring to Table I and Fig-

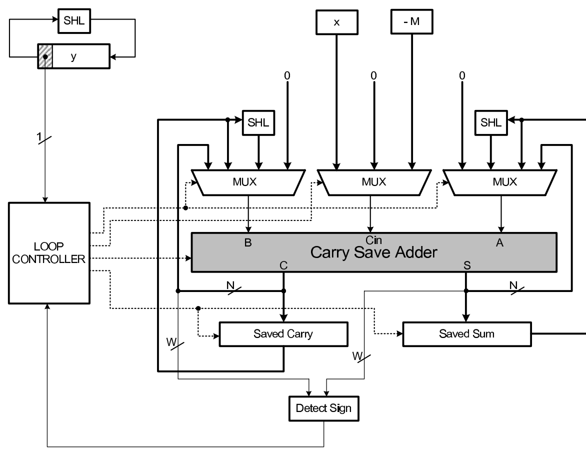

Fig. 2. Modified Interleaved Modular Multiplier

TABLE VIII
MAX. FREQUENCY FOR MODULAR MULTIPLIER

Precision in bits	Redundant interleaved	Modified interleaved
32	70.2 MHz	217.5 MHz
64	57.7 MHz	211.5 MHz
128	48.4 MHz	206.2 MHz
256	46.8 MHz	185.5 MHz
512	64.8 MHz	193.4 MHz
1024	69.4 MHz	181.0 MHz

ure 1, the pre-computed values depend on Y and M. Thus, the pre-computation overhead will be present at every iteration of the square and multiply algorithm. Our proposed architecture does not require any pre-computation. Table X summarizes the absolute time (pre-computation time + hardware time) per a single operation for different precisions. Timings are all listed in micro-seconds. Time overhead due to pre-computation was estimated using a software executed on a 32bit, 96 DMIPS RISC processor running at 85 MHz.

it's clear that the pre-computation (if exists) timing overhead can never be ignored, especially when it is required each time before multiplication. The overhead becomes superior to the HW time when performing modular exponentiation using square and multiply algorithm. Our proposed modular multiplier delivers the same output at few microseconds larger HW time, but without any need of software computation overhead. The total operation time is, thus, significantly improved.

VII. Conclusion

In this contribution, a modified version was introduced of the interleaved modular multiplier. The proposed architecture can achieve faster timings for most public key cryptosystems. The improved timing results comes at the cost of extra logic utilization. The architecture can be used in most asymmetric ciphers such as RSA, DSA, ECC, and DH key exchange. It has the advantage of avoiding look up tables used in all architecture presented in [6] and [4]. Look up tables were replaced by an improved version of carry save adders equipped with sign determination module. Implementation was done using VHDL description of the system. Simulation, synthesis, and post route simulation were done on a Xilinx Virtex 2 FPGA. Significant improvements in both operating frequency and absolute time per operation were achieved.

TABLE IX
DEVICE UTILIZATION FOR MODULAR MULTIPLIERS

Precision in bits	Redundant interleaved	Modified interleaved
32	0.70%	1.50%
64	1.40%	3.30%
128	2.70%	5.50%
256	5.40%	12.30%
512	8.00%	24.40%
1024	24%	39.30%

TABLE X
ABSOLUTE TIME (μs) PER AN OPERATION

	Redundant interleaved		Modified interleaved	
Bits	SW Time	HW	SW Time	HW
32	30.1	0.47	-	0.87
64	63.4	1.12	-	1.36
128	146	2.67	-	2.32
256	335	5.50	-	6.83
512	835	8.00	-	10.48
1024	2500	14.77	-	22.53

References

[1] Peter L. Montgomery, "Modular multiplication without trial division," in *Mathematics of Computation*. April 1985, vol. 44, pp. 519–521, American Mathematical Society.
[2] G. R. Blakley, "A computer algorithm for the product AB modulo M," *IEEE Transactions on Computers*, pp. 497 – 500, May 1983.
[3] D. Narh Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, "Efficient hardware architectures for modular multiplication on FPGAs," *International Conference on Field Programmable Logic and Applications*, pp. 539–542, 2005.
[4] V. Bunimov and M. Schimmler, "Area and time efficient modular multiplication of large integers," in *IEEE 14th International Conference on Application specific Systems, Architectures and Processors*, June 2003.
[5] Q.K. Kop and C.Y. Hung, "Fast algorithm for modular reduction," in *IEE Proceedings, Computers and Digital Techniques*, July 1998, vol. 145, pp. 265–271.
[6] David Narh Amanor, "Efficient hardware architectures for modular multiplication," M.S. thesis, University of Applied Sciences Offenburg, Germany, February 2005.