Faster Interleaved Modular Multiplier Based on Sign Detection

Mohamed A. Nassar, and Layla A. A. El-Sayed

Department of Computer and Systems Engineering,
Alexandria University, Alexandria, Egypt
eng.mohamedatif@gmail.com, labohadid@yahoo.com

Abstract

Data Security is the most important issue nowadays. A lot of cryptosystems are introduced to provide security. Public key cryptosystems are the most common cryptosystems used for securing data communication. The common drawback of applying such cryptosystems is the heavy computations which degrade performance of a system. Modular multiplication is the basic operation of common public key cryptosystems such as RSA, Diffie-Hellman key agreement (DH), ElGamal and ECC. Much research is now directed to reduce overall time consumed by modular multiplication operation. Abd-el-fatah et al. introduced an enhanced architecture for computing modular multiplication of two large numbers X and Y modulo given M. In this paper, a modification on that architecture is introduced. The proposed design computes modular multiplication by scanning two bits per iteration instead of one bit. The proposed design for 1024-bit precision reduced overall time by 38% compared to the design of Abd-el-fatah et al.

Keywords:

efficient architecture, carry-save adder, sign detection, sign estimation technique, modular multiplication, FPGA, RSA.

1. Introduction

Public key cryptosystems are implemented as coprocessors for accelerating cipher operation and reducing overhead of intensive computations from the main processor. Modular multiplication operation is the basic operation of the common public key cryptosystems such as RSA [10], Diffie-Hellman key agreement (DH) [7], ElGamal and ECC [1]. Speeding up modular multiplication operation will improve the efficiency of the whole public key cryptosystem. Much research is now directed to introduce enhanced architectures for modular multiplication operation to reduce the overall time of this operation.

Implementation of modular multiplication can be categorized into classical and interleaved. Classical implementation tries to solve the multiplication problem by computing multiplication and then reducing the result. Interleaved implementation interleaves multiplication with reduction at the same time. Reference [3] has detailed survey about classical and interleaved algorithms. An Example of interleaved algorithm is Montgomery modular algorithm [3, 14, 15].

In our proposed architecture, we introduce modification on architecture of interleaved modular multiplication based on improved sign detection [1]. One of benefits of modular multiplication based on sign detection is no need to save any value in a look up table. Another benefit is that the overall time is better than the architecture with look up table proposed by Amanor [2].

The rest of the paper is organized as follows: section 2 presents back ground and related work of interleaved Modular multiplication. Section 3 describes in details our proposed modification based on improved sign detection. It also proves the correctness of the new architecture. Section 4 summarizes results of the implementation compared with the architecture of Abd-el-fatah et al. [1]. A conclusion is given in Section 5.

2. Interleaved Modular Multiplication

The interleaved modular multiplication reduces the intermediate result (R) produced from multiplication to the range [0, M-1] at each iteration. Let X, Y, M and R be the multiplicand, the multiplier, the modulus and the result respectively, and let n be the number of bits in their binary representation. Table 1 shows the standard algorithm [2, 4].

Table 1. Standard interleaved modular multiplication [2,4].

```
Input: $X = (x_{n-1} \ldots x_0)$, $Y = (y_{n-1} \ldots y_0)$, $M = (m_{n-1} \ldots m_0)$
  where $0 \le X, Y < M$
Output: $R = X * Y \bmod M$
0-  R = 0;
1-  for i = n-1 downto 0
2-      R = R * 2;
3-      I = $y_i$ * X;
4-      R = R + I;
5-      if (R ≥ M) then R = R – M;
6-      if (R ≥ M) then R = R – M;
7-  end for;
```

The algorithm scans Y starting from the MSB. At each iteration, R is incremented twice per iteration at worst case, once by R (step (2)) and another conditional increment by X (steps (3) and (4)) which means R after performing step (4) will satisfy the following inequality: $0 \le R < 3M$. At the end of each iteration, the condition ($R \ge M$) has to be checked at most twice (steps (5) and (6)) so that R has to be less than M.

At each iteration, three main operations are performed; shift (step (2)), addition, subtraction (step (4), (5) and (6)) and comparison (steps (5) and (6)). Addition, subtraction and comparison are complex operations. For large numbers addition and subtraction, CSAs are used. Comparison (Steps (5) and (6)) needs a comparator between R and M. Worst case comparison will also results in a large propagation delay, as all bits from both numbers have to be compared [1]. The comparison needs both numbers to be in their final form, which conflicts with the CSA output form. For optimizing steps (5) and (6), two recent proposed algorithms and corresponding architectures [1, 4] were introduced as follows.

2.1 Redundant Interleaved Modular Multiplication

The authors in [4] simplify and significantly speed up the operations inside the loop. Table 2 shows the proposed algorithm. The authors speed up time required for modular multiplication by further exploiting pre-calculation of values into a lookup table. LookUp(i) function (steps (1), (8)) returns value of i^{th} entry in the look up table.

Table 2. Redundant interleaved modular multiplication algorithm[4]

```
Input: $X = (x_{n-1} \dots x_0)$, $Y = (y_{n-1} \dots y_0)$, $M = (m_{n-1} \dots m_0)$
        where $0 \le X, Y < M$
Output: $P = X * Y \bmod M$
Precomputing lookup table entries
0-  S = 0; C = 0
1-  A = LookUp(xn-1)
2-  for i=n-1 downto 0
3-      S = S mod 2n
4-      C = C mod 2n
5-      S = 2*S
6-      C= 2*C
7-      (S,C) = CSA(S,C,A)
8-      A=LookUp(2*(sn + 2*cn+1 + cn)+xi-1)
9-  end for
10-     P = (S + C) mod M
```

2.2 Modified Interleaved Modular Multiplication based on Sign Detection

Modified Interleaved Modular Multiplication based on Sign Detection (MIMM) algorithm replaced comparisons (steps (5), (6)) in standard algorithm mentioned in Table 1 by determining the sign of the quantity (R – M). The Sign determination is performed using the "Sign Detection" technique. This technique is explained in the next sub-section [1, 16].

2.2.1 Sign Detection

Previous work [16] proposed a technique for the sign estimation of a number. A window of only two bits taken from a (sum, carry) pair of a number is used in this technique. The window count starts from the MSB and moving downwards. The technique produces three possible results: positive (+ve), negative (-ve) and unsure (+/-ve). The unsure (+/-ve) result is reached when the number is either too large or too small [16]. Frequently reaching the unsure (+/-ve) result is the drawback of this technique [1].

AbdelFattah et al. [1] has enhanced the sign estimation technique, so that an exact sign can be determined for any number. A window of length (W) is taken from a number represented in (sum, carry) pair

The detect-sign technique DS(S, C) can be defined as follows [1]:

Let X be 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 P is positive.
4. Else if Temp != "11….11" then P is negative.
5. Else(Temp is all ones) request another carry-save addition, update (S, C) and Go to (1).

In case of the unsure (+/-ve) result (step (5)), S and C are fed back again to CSA, and CSA will produce a new permutation of S and C for the same result. The detailed proof of the correctness of detect-sign technique is provided in [1].

2.2.2 Algorithm

The proposed algorithm is shown in Table 3. DS(R - M) function (steps (5) and (6)) applies sign detection technique for the number (R - M).

Table 3. Modified interleaved modular multiplication [1].

```
Input: $X = (x_{n-1} \ldots x_0)$, $Y = (y_{n-1} \ldots y_0)$, $M = (m_{n-1} \ldots m_0)$
       where $0 \leq X,Y < M$
Output: $R = X * Y \bmod M$
0- R = 0
1- for i = n-1 downto 0
2-     R = R * 2
3-     I = yi * X
4-     R = R + I;
5-     if DS(R - M)=+ve  R = R – M
6-     if DS(R - M)=+ve  R = R – M
7- end for
```

2.2.2 The Architecture

Fig. 1 shows the architecture of MIMM [1]. The main modules of the architectures are:

1. Carry-Save Adder;
2. Detect-Sign Module (DS);
3. Controller;
4. Registers (SavedSum, SavedCarry);
5. Multiplexers (MUX_1, MUX_2, MUX_3).

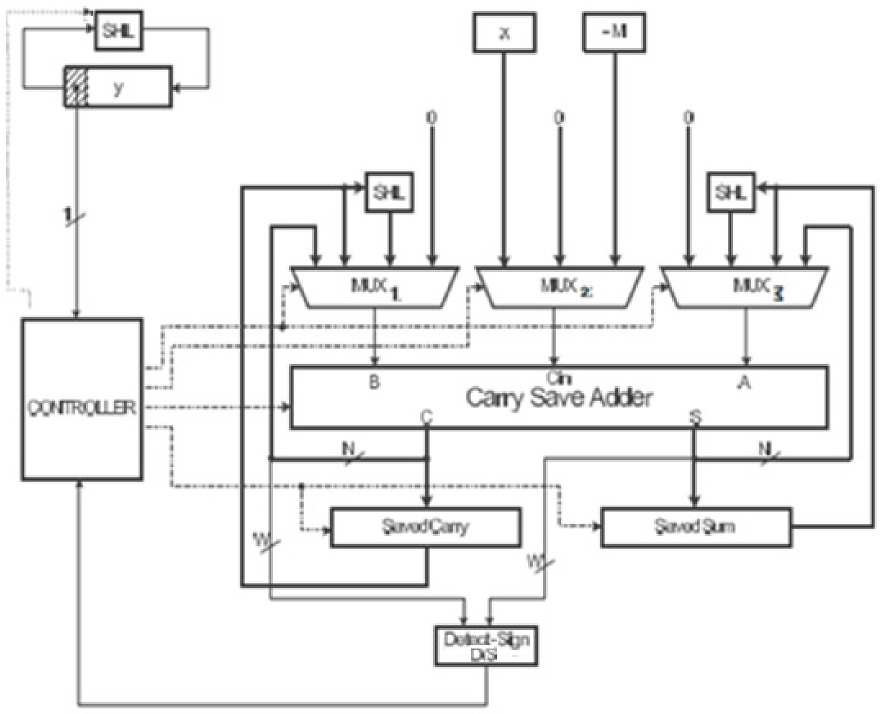

Fig. 1. Modified interleaved modular multiplier [1].

MIMM computes modular multiplication without a pre-computational phase, or predefined sets for moduli [1, 9].

3. Faster Interleaved Modular Multiplier Based on Sign Detection

The idea is to enhance the performance of MIMM by scanning two bits instead of one bit at each iteration. Let X, Y, M and R be the multiplicand, the multiplier, the modulus and the result of modular multiplication respectively (R = X * Y mod M), and let n be an even number of bits in their binary representation as follows.

$$X = \sum_{i=0}^{n-1} x_i\, 2^i\ ,\ \mathrm{Y} = \sum_{\mathrm{i}=0}^{\mathrm{n}-1} \mathrm{y_i}\, 2^{\mathrm{i}}\ , M = \sum_{i=0}^{n-1} m_i\, 2^i$$

The algorithm is shown in Table 4. Steps (5), (6) and (7) are performed at each iteration to ensure that intermediate result R satisfies the following inequality: $0 \leq R < M$.

Table 4. Faster interleaved modular multiplication based on sign detection algorithm.

Input: *$X = (x_{n-1} \ldots. x_0)$, $Y = (y_{n-1} \ldots\ldots y_0)$, $M = (m_{n-1} \ldots.. m_0)$*

where $0 \le X, Y < M$ and $n \bmod 2 = 0$

Output: *$R = X*Y \bmod M$*

```
0- R = 0
1- for i = n/2 - 1 downto 0
2-         R = R * 4
3-         I = [$y_{n-1}$ $y_{n-2}$] * X
4-         R = R + I
5-         if  DS(R - 4M)= +ve  R = R - 4M
6-         if  DS(R - 2M)= +ve  R = R - 2M
7-         if  DS(R - M)=  +ve  R= R - M
8-         Y = Y << 2
9- end for
```

3.1 Proof of Correctness

It is required to prove that the final result R is correct, i.e. $0 \le R < M$. First we prove that the upper bound of the intermediate result R (step (4) of the algorithm mentioned in Table 4) is less than 7M as follows:

For the first iteration:

- At step (4), $R = [y_{n-1}\ y_{n-2}] * X < 3M$.
- Steps (5), (6) and (7) guarantee that M satisfies the following inequality: $0 \le R < M$.

For the i^{th} iteration where $i > 0$:

- At step (4), $R = R * 4 + [y_{n-1}\ y_{n-2}] * X < 7M$.
- Steps (5), (6) and (7) guarantee that M satisfies the following inequality: $0 \le R < M$.

Table 5 shows all possible ranges for intermediate result R (Step (4)) and the corresponding action to reduce it.

Table 5. All possible ranges for R and the corresponding actions.

Range	*Step(s) where subtraction(s) is(are) confirmed*
$0 \leq R < M$	No action
$M \leq R < 2M$	Step 7 $(R - M)$
$2M \leq R < 3M$	Step 6 $(R = R - 2M)$
$3M \leq R < 4M$	Steps 6 and 7 $(R = R - 3M)$
$4M \leq R < 5M$	Step 5 $(R = R - 4M)$
$5M \leq R < 6M$	Steps 5 and 7 $(R = R - 5M)$
$6M \leq R < 7M$	Steps 5 and 6 $(R = R - 6M)$

3.2 The Architecture

As shown in Fig.2, the main modules found in the proposed architecture are:

1. Carry-Save Adders (CSA_1 and CSA_2);
2. Detect-Sign module (DS);
3. Registers (SavedSum, SavedCarry);
4. Two-bit left shifters for shifting Y, 2*SavedSum, 2*SavedCarry, and computing -4M;
5. One-bit left shifters for computing 2X and -2M;
6. Multiplexers (MUX_1, MUX_2, MUX_3);
7. The Controller.

Our architecture is an adaptation of MIMM architecture [1]. The main differences between our architecture and MIMM architecture shown in Fig. 1 are:

1. Another CSA called CSA_2 was added to compute 3*X;
2. Two-bit left shifters are used to shift the input Y and compute -4*M;
3. MUX_1 and MUX_2 have different number of inputs as shown in Fig 2;
4. The controller module is more complex than that of MIMM.

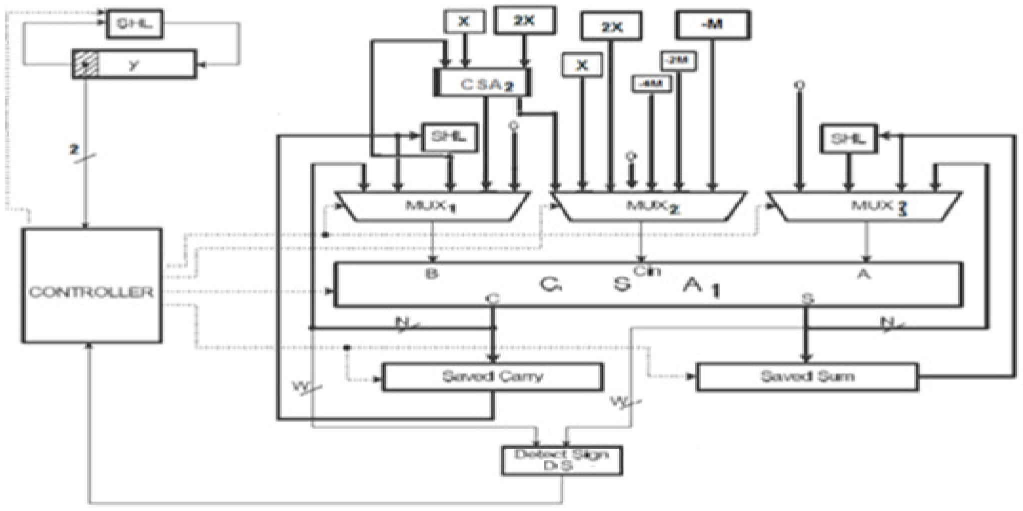

Fig. 2. Architecture of Faster modular multiplication based on sign detection

4. Implementations and Results

Synthesis using Xilinx Synthesis Tool (XST) was performed on a Xilinx FPGA Vertex XC4VFX12 [6, 11, 12]. The synthesis of the VHDL implementations of the architectures was performed with speed optimization goal [13]. Architectures are verified using software library called MIRACL [5] as follows: using MIRACL, three random numbers are generated (A, B and M where A and B are less than M) and Result R (R = A * B mod M) is computed. A, B and M are loaded on different architectures and R′ (output result from each architectures) is compared with corresponding R.

The synthesis tool (XST) generated the design reports for each of the implemented multipliers. The minimum clock period, and total number of slices used for each implementation are tabulated and graphically analyzed. Number of slices and Maximum frequency used for the different architectures are shown in Table 6 and Table 7 respectively.

Table 6. Number of slices used for the different architectures.

Precision	*MIMM*	*FIMM*	*FIMM area increase*
32	222	373	68%
64	415	612	47%
128	803	1163	45%
256	1584	2277	44%
512	3117	4493	44%
1024	6221	8930	44%

Table 7. Maximum frequency for the different architectures.

Precision	*MIMM(MHZ)*	*FIMM (MHZ)*
32	521.38	371.47
64	490.44	333.67
128	491.64	311.24
256	404.69	263.37
512	344.23	311.72
1024	308.83	325.84

From Table 6, the proposed architecture has on average 48% more area than MIMM; this is due to the more logic in controller and adding other CSAs.

For different precisions, average improvements were calculated for txt data type. Table 8 and Fig. 3 summarize average overall operation time for the different architectures. Also Table 8 summarizes average speed up of our architectures for different precision.

Table 8. Average overall operation time consumed by the different architectures for txt data type.

Precision	*Average overall operation time*		
	MIMM(ns)	*FIMM (ns)*	*Relative speed up gain for FIMM*
32	196.99	218.74	-10%
64	425.73	489.81	-13%
128	792.41	921.17	-16%
256	2074.79	2491.75	-17%
512	4931.96	4301.25	15%
1024	11138.68	8091.18	38%

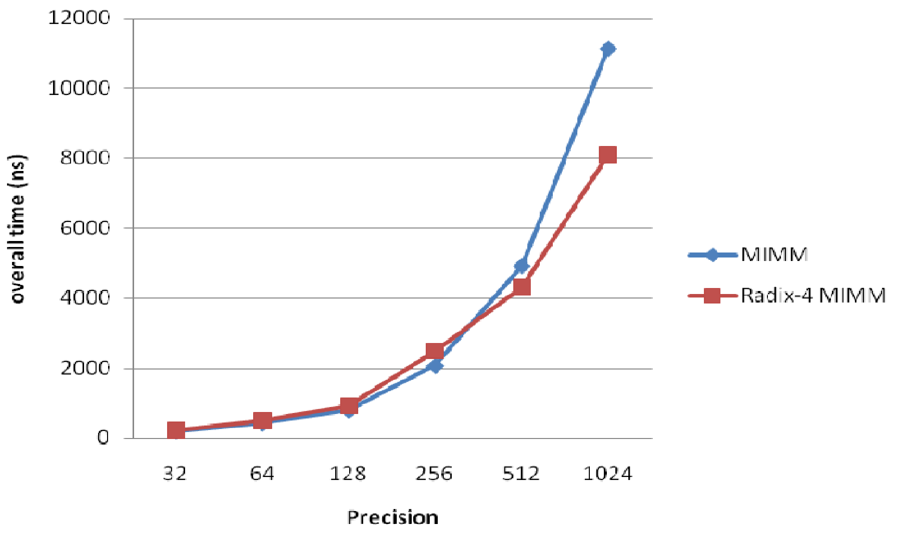

Fig. 3. Average overall operation time consumed by the different architecture for txt data type.

5. Conclusion

In this paper, a new proposed interleaved modular multiplication algorithm and the corresponding architectures (FIMM) were introduced. The proposed architecture is based on sign detection technique which is responsible for determining the sign of a number represented in a (Sum, Carry) pair. FIMM architecture improved substantially the time requirement compared with MIMM architecture when the operands size is more than 256 bits. The relative speedup gain of FIMM architecture compared with MIMM is up to 38% for 1024-bit precision. Our architecture has not a pre-computational phase or any restrictions on moduli which make our architectures more efficient and more flexible than existing architectures. The architectures were implemented for different precisions 32, 64, 128, 256, 512 and 1024 bits. VHDL was used to implement different architectures. Simulations were performed on a Xilinx FPGA XC4VFX12.

References

[1] AbdelFattah et al., “Efficient Implementation of Modular Multiplication on FPGAs Based on Sign Detection,” Proc. 4th International Design and Test Workshop (IDT), pp. 1 - 6, February 2010.

[2] D. Narh Amanor, “Efficient Hardware Architectures for Modular Multiplication,” M.S. thesis, University of Applied Sciences Offenburg, Germany, February 2005.

[3] Nadia Nedjah, “A Review of Modular Multiplication Methods and Respective Hardware Implementations,” Proc. Informatica 30, pp. 111-129, 2006.

[4] D. Narh Amanor, C. Paar, J. Pelzl, V. Bunimov, and M. Schimmler, “Efficient hardware architectures for modular multiplication on FPGAs,” Proc. International Conference on Field Programmable Logic and Applications, pp. 539–542, 2005.

[5] MIRACL, “Multi-precision Integer and Rational Arithmetic C/C++ Library,” http://www.shamus.ie/, last referenced January 20, 2011.

[6] Xilinx, Inc. Foundation Series Software, http://www.xilinx.com, last referenced January 20, 2011.

[7] Diffie, and Hellman, "New Directions in Cryptography," IEEE Transactions on Information Theory, vol. IT-22, no. 6, pp. 644-654, November 1976.

[8] Paniandi, "A Hardware Implementation of Rivest-Shamir-Adleman Co-processor or Resource Constrained Embedded Systems," M.S. thesis, University of Technology Malaysia, April 2008.

[9] Miroslav Knezevic, "Faster Interleaved Modular Multiplication Based on Barrett and Montgomery Reduction Methods," IEEE Transactions on Computers, vol. 59, no. 12, pp. 1715-1721, December 2010.

[10] R.L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems," Communications of the ACM, vol. 21, no. 2, February 1978.

[11] Timing Constraints User Guide, http://www.xilinx.com/support/ documentation/sw_manuals/xilinx12_3/ug612.pdf, last referenced March 20, 2011.

[12] Virtex-5 FPGA User Guide, http://www.xilinx.com/support/documentation/ user_guides /ug190. pdf, last referenced March 20, 2011.

[13] VHDL Reference Manual, http://www.usna.edu/EE/ee462/ manuals/vhdl_ref.pdf, last referenced March 20, 2011.

[14] Alexandre F. Tenca, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm," IEE Transactions on Computer, vol. 52, no. 9, September 2003.

[15] Nathaniel Pinckney, "Parallelized Radix-4 Scalable Montgomery Multipliers," Journal of Integrated Circuits and Systems, pp. 28 30, 2008.

[16] Q.K. Kop and C.Y. Hung, "Fast algorithm for modular reduction," in IEE Proceedings, Computers and Digital Techniques, vol. 145, pp. 265–271, July 1998.