



A Survey on Hardware Architectures for Montgomery Modular Multiplication Algorithm

Ambika R¹, Hamsavahini R²
Electronics & Communication Engg
BMS Institute of Technology
INDIA

Abstract: The motivation for studying high speed and space efficient algorithms for modular multiplications comes from their applications in public – key cryptography. The Montgomery multiplication algorithm speeds up the modular multiplications and squaring required for the exponentiation. This paper presents new architectures for the computation of modular multiplication and exponentiation using Montgomery multiplication (MM) algorithm. Montgomery modular multiplication (MMM) is one of the fundamental operations used in cryptographic algorithms such as RSA, Diffie-Hellman key distribution and Elliptic Curve cryptosystems. In this paper we compare the new hardware architectures that are able to perform Montgomery multiplication.

Keywords: Architecture, Cryptography, Field Programmable Gate Array, Montgomery multiplication, MMR2MM algorithm, MMR4MM algorithm, RSA, Scalable multiplier

I. Introduction

Due to the rapid increase in data communications and internet services, such as electronic commerce, fundamental security requirements for protecting sensitive data during electronic dissemination has become an important concern. Public-key cryptosystems are important as they provide confidentiality, authentication and data integrity and nonrepudiation [1,2]. Many systems utilize public-key cryptography to provide such security services, and Rivest, Shamir, and Adleman (RSA) [3] is one of the most widely adopted public key algorithms at present. After the introduction of the RSA algorithm in 1978, high speed and space efficient hardware architectures for modular multiplication have been a subject of constant interest for more than 30 years. With the introduction, RSA requires repeated modular multiplications to accomplish the computations of modular exponentiation. Montgomery multiplication is the basic operation of the modular exponentiation which is required in the RSA public cryptosystem.

Modular exponentiation with large modulus is usually accomplished by performing reiterated modular multiplications which is an operation that consumes a considerable amount of time. As a result, the speed of an RSA cryptosystem will be entirely dependent on the throughput rate of modular multiplication and the number of performed modular multiplications. To speed up the method of modular multiplication, the Montgomery's multiplication algorithm [4] is generally used.

In this paper we focus on the comparison of optimized hardware architectures for Multiple-Word Radix-2 Montgomery Multiplication (MWR2MM) and Multiple-Word Radix-4 Montgomery Multiplication (MWR4MM) algorithms in order to minimize the number of clock cycles required to compute an n-bit precision Montgomery Multiplication. Section 2 contains introduction to Montgomery Multiplication, Multiple word radix 2 Montgomery algorithm and optimized architecture MWR2MM algorithm is presented in section 3. High radix version for Montgomery Multiplication is presented in section 4. In section 5 comparisons of algorithms from conceptual point of view is presented and finally in section 6 the summary and conclusion of the paper.

II. Montgomery Modular Multiplication Algorithm

Let $M > 0$ be an odd integer. In many cryptosystems, such as RSA, computing $XY \pmod{M}$ is a crucial operation. The reduction of $XY \pmod{M}$ is a more time-consuming step than the multiplication XY without reduction. In [4], Montgomery introduced a method for calculating products $\pmod{M}$ without the costly reduction $\pmod{M}$, since then known as Montgomery multiplication. Montgomery multiplication of X and $Y \pmod{M}$, denoted by $MP(X; Y; M)$, is defined as $XY \cdot 2^{-n} \pmod{M}$ for some fixed integer n .

Algorithm1:Radix 2 Montgomery Multiplication

input : odd M , $n = \lceil \log_2 M \rceil + 1$, $X = \sum_{i=0}^{n-1} x_i 2^i$ with
 $0 \leq X, Y < M$
output : $\bar{Z} = MP(X, Y, M) \equiv X.Y.2^{-n} \pmod{M}$, $0 \leq Z < M$
1.1 $s[0] = 0$;
1.2 for $i = 0$ to $n - 1$ do
1.3 $q_i = (x_i.Y_0) \oplus S[i]_0$;
1.4 $S[i + 1] = (S[i] + x_i.Y + q_i.M)/2$;
1.5 if $S[n] > M$ then
1.6 $S[n] = S[n] - M$;
1.7 return $Z = S[n]$;

The operations during this formula are simply enforced in hardware. However, these operations are performed with full exactness operands and, during this sense, they need an intrinsic limitation. Once a hardware supported algorithm is defined for n bits, it cannot work with more bits. To get rid of this limitation and keep the straightforward operations of the radix-2 formula, an alternative algorithm is projected, as explained in the next section.

III. Multiple-Word Radix-2 Montgomery Multiplication (MWR2MM)

In Algorithm 2[10], the data Y (multiplicand) is scanned word-by-word, and therefore the data X is scanned in bit by bit. The data length is n bits, and therefore the word length is w bits. $E = \lceil n+1/w \rceil$ words are needed to store S since its vary from $[0, 2M-1]$. The original M and Y are extended by an additional bit of zero because the most significant bit is represented as vectors.

Algorithm2: Multiple word Radix-2 Montgomery multiplication Algorithm

Input : odd M , $n = \lceil \log_2 M \rceil + 1$, wordsize w , $e = \left\lceil \frac{n+1}{w} \right\rceil$,
 $X = \sum_{i=0}^{n-1} x_i.2^i$, $Y = \sum_{j=0}^{e-1} Y^{(j)}.2^{w.j}$,
 $M = \sum_{j=0}^{e-1} M^{(j)}.2^{w.j}$, with $0 \leq X, Y < M$
Output : $Z = \sum_{j=0}^{e-1} S^{(j)}.2^{w.j} = MP(X, Y, M) \equiv X.Y.2^{-n} \pmod{M}$, $0 \leq Z < 2M$
/ * Initialize all words of S * /
2.1 $S = 0$;
2.2 for $i = 0$ to $n - 1$ do
2.3 $q_i = (x_i.Y_0^{(0)}) \oplus S_0^{(0)}$;
2.4 $(C^{(1)}, S^{(0)}) = x_i.Y_{(0)}^{(0)} + q_i.M^{(0)} + S^{(0)}$;
2.5 for $j = 1$ to step 1 do
2.6 $C^{(j+1)}, S^{(j)} = C^{(j)} + x_i.Y^{(j)} + q_i.M^{(j)} + S^{(j)}$;
2.7 $S^{(j-1)} = (S_0^{(j)}, S_{w-(1 \dots 1)}^{(j-1)})$;
2.8 $S^{(e)} = 0$;
2.9 return $Z = S$;

The data dependency graph of the hardware implementation for the MWR2MM algorithm by Tenca and Koc, is shown in Fig. 1. Every circle within the graph represents an atomic computation and is labeled in line with the sort of action performed. Task A consists of performing the operations of lines 2.3 and 2.4 in Algorithm2. Task B corresponds to perform the operations of lines 2.6 and 2.7 in Algorithm2.

The data dependencies among the operations among j loop make it not possible to execute the steps in an exceedingly single iteration of j loop in parallel. However, correspondence is feasible among executions of various iterations of i loop. In [6], Tenca and Koc, urged that every column within the graph could also be computed by a separate processing Element (PE), and therefore the data generated from one PE could also be passed into another PE in an exceedingly pipelined fashion. Following this methodology, all atomic computations pictured by circles within the same row are often processed at the same time. The processing of each column takes $e + 1$ clock cycles (one clock cycle for Task A, e clock cycles for Task B). Because there is a delay of two clock cycles between the processing of a column for x_i and the processing of a column for x_{i+1} , the minimum computation time T (in clock cycles) is $T = 2n + e - 1$ given $P_{max} = \lceil (e+1)/2 \rceil$ PEs are implemented to work in parallel. In this configuration, after $e + 1$ clock cycles, PE #0 switches from executing column 0 to executing column P_{max} . After another two clock cycles, PE #1 switches from executing column 1 to executing column $P_{max}+1$, etc. The opportunity of improving the implementation performance of algorithm 2 is to reduce the delay between the processing of 2 succeeding iterations of i loop from 2 clock cycles to at least one clock cycle. The two-clock-cycle delay comes from the proper right shift (division by two) in Algorithm 1 and 2. Take the initial 2 PEs in Fig1 as an example. These 2 PEs reckon the S words within the initial 2 columns. Starting from clock #0, PE #1 must wait for 2 clock cycles before it starts the computation of $S(0)$ ($i=1$) in the clock cycle #2.

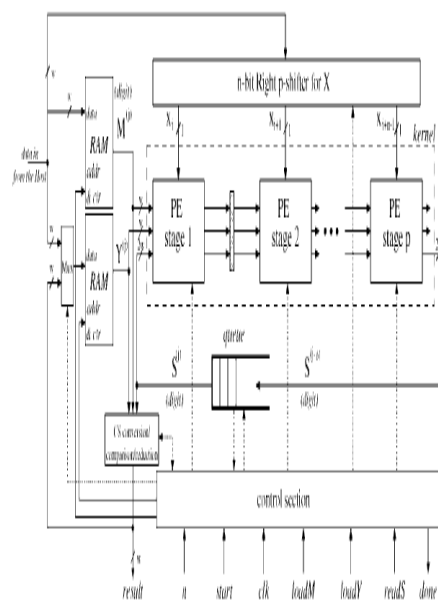
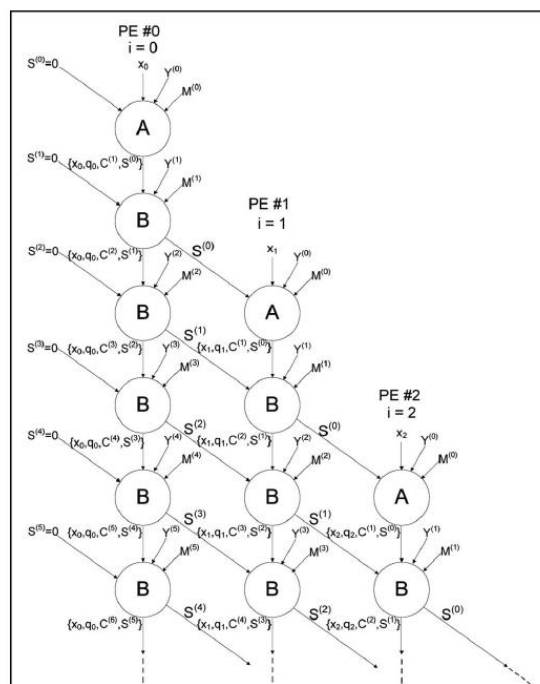
A. Scalable architecture for MWR2MM

A pipelined organization for the system is shown in Fig. 4. The pipeline itself was named kernel in the figure and it is composed of PEs. The other blocks represent memory, data conversion, and control unit. Each processing element in the pipeline relays the received words to the next downstream unit. All paths are w -bits wide, except for the x_i inputs (only 1 bit). The kernel itself does not limit the final computation precision. If more precision is required, it is only necessary to feed more words. The final and intermediate results are stored in the queue. Gray boxes indicate registers. The control block function can be inferred from the algorithm description that was provided, combined with other data manipulation tasks that must be done to transfer data between the multiplier and the host system.

B. Energy-Efficient Montgomery Modular Multiplication Algorithm

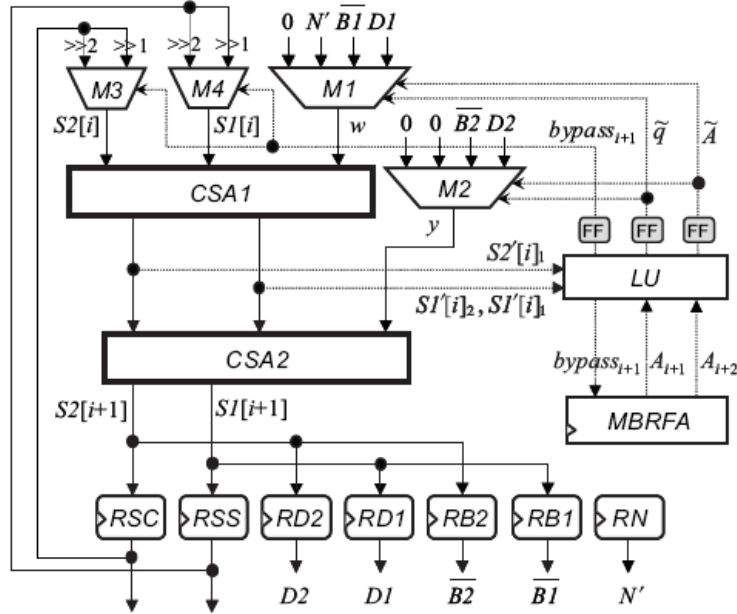
To speed up the encryption/decryption method, several high-speed Montgomery standard multiplication algorithms and hardware architectures use carry-save addition to avoid the carry propagation at every addition operation of the add-shift loop. In this paper, ShiananJiun and Huan[9] proposed an energy-efficient algorithm and its corresponding architecture to not only reduce the energy consumption but also further enhance the throughput of Montgomery modular multipliers.

Fig1: Data dependency graph of the original architecture of MWR2MM algorithm [2] Fig. 2. Pipelined organization for the multiplier.



The projected design is capable of bypassing the superfluous carry-save addition and register write operations, resulting in less energy consumption and better throughput. They also modify the barrel register full adder (BRFA) in order that the gated clock style technique are often applied to considerably reduce the energy consumption of storage components in BRFA.. Experimental results show that the proposed approaches n achieve up to 60% energy saving and 24.6% throughput improvement for 1024-bit Montgomery multiplier.

Fig 3:Block diagram of proposed MMM42 multiplier.



IV. Multiple-Word Radix-4 Montgomery Multiplication(MWR4MM)

Instead of scanning one little bit of X on every time, many bits of X are often scanned along for high-radix cases. Let k bits of X are scanned at just once, 2k branches should be covered at the same time to maximize the performance. Considering the value of 2k increases exponentially as k increments, the design becomes impractical on the far side radix-4. Following constant definitions relating to words as in Algorithm 2, the radix-4 version of Montgomery multiplication is shown in Algorithm 3. 2 bits of X are scanned in one step this point rather than one bit as in Algorithm 2. Whereas reaching the maximal parallelism, the radix-4 version style takes $n/2 + e - 1$ clock cycles to process n-bit Montgomery multiplication.

Algorithm3: Multiple word Radix-4 Montgomery multiplication Algorithm

$$\begin{aligned}
 & \text{input : odd } M, n = \lceil \log_2 M \rceil + 1, \text{ wordsize } w = \left\lceil \frac{n + 1}{w} \right\rceil, \\
 & X = \sum_{i=0}^{\left\lceil \frac{n}{2} \right\rceil - 1} x^i . 4^i \\
 & Y = \sum_{j=0}^{e-1} Y^{(j)} . 2^{w \cdot j}, \\
 & M = \sum_{j=0}^{e-1} M^{(j)} . 2^{w \cdot j}, \text{ with } 0 \leq X, Y < M \\
 & \text{Output : } Z = \sum_{j=0}^{e-1} S^{(j)} . 2^{w \cdot j} = MP(X, Y, M) \equiv X.Y.2^{-n} \\
 & (\text{mod } M), 0 \leq Z < 2M
 \end{aligned}$$

```

/ * Initialize all words of S * /
3.1 S = 0;
3.2 for i = 0 to n - 1 do
3.3    $q_i = \text{Func}(S_{1..0}^{(0)}, x^{(i)}, Y_{1..0}^{(0)}, M_{1..0}^{(0)});$ 
3.4    $(C^{(1)}, S^{(0)}) = S^{(0)} + x^{(i)} \cdot Y^{(0)} + q^{(i)} \cdot M^{(0)};$ 
3.5   for j = 1 to e - 1 step 1 do
3.6      $C^{(j+1)}, S^{(j)} = C^{(j)} + S^{(j)} + x_i \cdot Y^{(j)} + q^{(i)} \cdot M^{(j)};$ 
3.7      $S^{(j-1)} = (S_{1..0}^{(j)}, S_{w-(1..2)}^{(j-1)});$ 
3.8    $S^{(e-1)} = (C_{1..0}^{(e)}, S_{w-1..2}^{(e-1)});$ 
3.9 return Z = S;

```

V. Comparison

In this section, we compare the major forms of architectures for Montgomery multiplication from the point of view of the amount of PEs and latency in clock cycles. Within the design by Tenca and Koc., the amount of PEs will vary between one and $P_{\max} = (e+1/2)$. Larger the amount of PEs, the smaller the latency, however the larger the circuit space. This feature permits the designer to decide on the best possible trade-off between these 2 necessities. The design by Harris et. al [7] has the similar scalability because the original design by Tenca and Koc, [6]. Rather than performing right shift of the intermediate $S(j)$ values, their design left shifts the Y and M to avoid the data dependency between $S^{(j)}$ and $S^{(j-1)}$.

VI. Conclusion

In this paper, we have compared the hardware architectures for Montgomery multiplication. These architectures are based on the new plan for enhancing parallelism by pre computing partial results. The architectures reduce circuit latency by a factor of 2, from $2n+e-1$ clock cycles to $n+1-e$ clock cycles. with a negligible penalty in terms of the minimum period of clock. In [8] systolic implementation of the Montgomery multiplier are done for full precision of the operands. A systolic multiplier for $n=512$ bits consumes about 50K gates and performs the operation in approximately $2n$ clock cycle. But in paper [8] with the configuration of $w=8$ and $p=40$ uses an area of 28 K and computes the multiplication in 1103 clock cycles. In Shiann, et.al [9] presented an efficient algorithm and its corresponding architecture to reduce the energy consumption and enhance the throughput of Montgomery modular multipliers simultaneously. Moreover, we modified the structure of BRFA and adopted the gated clock design technique to further reduce the energy consumption of Montgomery modular multipliers. Experimental results showed that the proposed approaches are indeed capable of reducing the energy consumption of the Montgomery multipliers. In future, we will try to heighten the occurring probability of superfluous operation bypassing to further reduce the energy consumption and enhance the throughput of modular multiplication.

References

- [1] Stallings, W.: 'Network and Internetwork Security Principles and Practice' (Prentice Hall, New York, USA, 1995)
- [2] Menezes, A., Oorschot, P., and Vanstone, S.: 'Handbook of Applied Cryptography' (CRC Press, Florida, USA, 1997)
- [3] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signature and public-key cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
- [4] P.L. Montgomery, "Modular multiplication without trial division," *Math.Comput.*, vol.44, No. 170, pp. 519-521, Apr.1985.
- [5] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electron. Lett.*, vol. 35, no. 21, pp. 1831–1832, Oct. 1999.
- [6] A.F. Tenca and C. K. Koc., "A Scalable Architecture for Montgomery Multiplication," *Proc. First Int'l Workshop Cryptographic Hardware and Embedded Systems (CHES '99)*, pp. 94-108, 1999.
- [7] D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, "An Improved Unified Scalable Radix-2 Montgomery Multiplier," *Proc. 17th IEEE Symp. Computer Arithmetic (ARITH)*, pp. 172-178, June 2005.
- [8] Alexandre F. Tenca, Member, IEEE, and C. etin K. Koc., Senior Member, IEEE, "A Scalable Architecture for Modular Multiplication Based on Montgomery's Algorithm" *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 52, NO. 9, SEPTEMBER 2003 1215
- [9] Shiann-Rong Kuang, Member, IEEE, Jiun-Ping Wang, Kai-Cheng Chang, and Huan-Wei Hsu "Energy-Efficient High-Throughput Montgomery Modular Multipliers for RSA Cryptosystems" *IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS*
- [10] Miaoqing Huang,, Kris Gaj, and Tarek El-Ghazawi, "New Hardware Architectures for Montgomery Modular Multiplication Algorithm" *IEEE TRANSACTIONS ON COMPUTERS*, VOL. 60, NO. 7, JULY 2011
- [11] Kuang, S.-R.; Wang, J.-P.; Chang, K.-C.; Hsu, H.-W., "Energy-Efficient High-Throughput Montgomery Modular Multipliers for RSA Cryptosystems," *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, vol.PP, no.99

Acknowledgments

The authors would like to express their sincere gratitude to the Management and staff, BMS Institute of Technology for providing the facilities.