International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

COPY RIGHT

IJIEMR Transactions, online available on 3rd Nov 2017. Link
:http://www.ijiemr.org/downloads.php?vol=Volume-6&issue=ISSUE-10

Title: **VLSI IMPLEMENTATION OF HIGH PERFORMANCE MONTGOMERY MODULAR MULTIPLICATION FOR CRYPTO GRAPHICAL APPLICATION**

Volume 06, Issue 10, Pages: 7 – 14.
Paper Authors

ANGIREKULA ANUSHA, KOTESHWAR RAO
GANAPATHI ENGINEERING COLLEGE

USE THIS BARCODE TO ACCESS YOUR ONLINE PAPER

To Secure Your Paper As Per UGC Guidelines We Are Providing A Electronic Bar Code

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

VLSI IMPLEMENTATION OF HIGH PERFORMANCE MONTGOMERY MODULAR MULTIPLICATION FOR CRYPTO GRAPHICAL APPLICATION

[1]ANGIREKULA ANUSHA, [2]KOTESHWAR RAO

PG Student, Dept. of ECE,Ganapathi Engneering COLLEGE

Assistant Professor, Dept. of ECE,Ganapathi Engneering COLLEGE

ABSTRACT--- The multiplier receives and outputs the data with binary representation and uses only one-level Carry Save Adder (CSA) to avoid the carry propagation at each addition operation. This CSA is also used to perform operand pre computation and format conversion from the carry save format to the binary representation, leading to a low hardware cost and short critical path delay at the expense of extra clock cycles for completing one modular multiplication. To overcome the weakness, a Configurable CSA (CCSA), which could be one full-adder or two serial half-adders, is proposed to reduce the extra clock cycles for operand pre computation and format conversion by half. The mechanism that can detect and skip the unnecessary carry-save addition operations in the one-level CCSA architecture while maintaining the short critical path delay is developed. The extra clock cycles for operand pre computation and format conversion can be hidden and high throughput can be obtained. AES is based on a design principle known as a substitution-permutation network, combination of both substitution and permutation, and is fast in both software and hardware. AES does not use a Feistel network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits.AES operates on a 4×4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field.

Keywords--Carry-save addition, low-cost architecture, Montgomery modular multiplier, public-key cryptosystem

1. INTRODUCTION

In Many public-key cryptosystems [1]–[3], modular multiplication (MM) with large integers is the most critical and time-consuming operation. Therefore, numerous algorithms and hardware implementation have been presented to carry out the MM more quickly, and Montgomery's algorithm is one of the most well-known MM algorithms. Montgomery's algorithm [4] determines the quotient only depending on

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

the least significant digit of operands and replaces the complicated division in conventional MM with a series of shifting modular additions to produce $S = A \times B \times R{-}1 \pmod N$, where N is the k-bit modulus, $R{-}1$ is the inverse of R modulo N, and $R = 2k \bmod N$. As a result, it can be easily implemented into VLSI circuits to speed up the encryption/decryption process. However, the three-operand addition in the iteration loop of Montgomery's requires long carry propagation for large operands in binary representation. To solve this problem, several approachesof based on carry-save addition were proposed to achieve a significant speedup of Montgomery MM. Based on the representation of input and output operands, these approaches can be roughly divided into semi-carry-save (SCS) strategy and full carry-save (FCS) strategy. In the SCS strategy [5]–[8], the input and output operands (i.e., *A*, *B*, *N*, and *S)* of the Montgomery MM are represented in binary, but intermediate results of shifting modular additions are kept in the carry-save format to avoid the carry propagation. However, the format conversion from the carry-save format of the final modular product into its binary representation is needed at the end of each MM. This conversion can be accomplished by an extra carry propagation adder (CPA) [5] or reusing the carry-save adder (CSA) architecture [8] iteratively. Contrary to the SCS strategy, the FCS strategy [9], [10] maintains the input and output operands *A*, *B*, and *S* in the carry-save format, denoted as (*AS*, *AC*), (*BS*, *BC*), and (*SS*, *SC*), respectively, to avoid the format conversion, leading to fewer clock cycles for completing a MM. Nevertheless, this strategy implies that the number of operands will increase and that more CSAs and registers for dealing with these operands are required. Therefore, the FCS-based Montgomery modular multipliers possibly have higher hardware complexity and longer critical path than the SCS-based multipliers.

2.PREVIOUSLY PROPOSED ARCHITUCTURE

2.1. Montgomery Modular Multiplier

In propose a new SCS-based Montgomery MM algorithm to reduce the critical path delay of Montgomery multiplier. In addition, the drawback of more clock cycles for completing one multiplication is also improved while maintaining the advantages of short critical path delay and low hardware complexity [2].

2.2. Critical Path Delay Reduction

The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM-2. That is pre compute $D = B + N$ and reuse the one-level CSA architecture to perform $B+N$ and the format conversion. Figure.1 shows the modified SCS-based Montgomery multiplication (MSCS-MM) algorithm and possible hardware architecture, respectively [3].

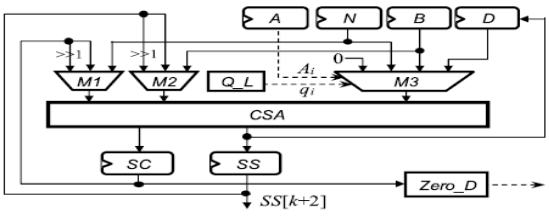

FIG.1. Diagram of Montgomery Modular Multiplier

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

The Zero_D circuit is used to detect whether SC is equal to zero, which can be accomplished using one NOR operation. The Q_L circuit decides the qi value. The carry propagation addition operations of B + N and the format conversion are performed by the one-level CSA architecture of the MSCS-MM multiplier through repeatedly executing the carry-save addition (SS, SC) = SS + SC + 0 until SC = 0.In addition, we also pre compute Ai and qi in iteration i−1 (this will be explained more clearly in Section III-C) so that they can be used to immediately select the desired input operand from 0, N, B, and D through the multiplexer M3 in iteration I [5] .Therefore, the critical path delay of the MSCS-MM multiplier can be reduced into *T*MUX4 + *T*FA. However, in addition to performing the three-input carry-save additions $k + 2$ times, many extra clock cycles are required to perform $B + N$ and the format conversion via the one-level CSA architecture because they must be performed once in every MM. Furthermore, the extra clock cycles for performing $B+N$ and the format conversion through repeatedly executing the carry-save addition $(SS, SC) = SS+SC+0$ are dependent on the longest carry propagation chain in $SS + SC$. If $SS = 111\ldots1112$ and $SC = 000\ldots0012$, the one-level CSA architecture needs k clock cycles to complete $SS + SC$. That is, $3k$ clock cycles in the worst case are required for completing one MM. Thus, it is critical to reduce the required clock cycles of the MSCS-MM multiplier [1].

2.3. Clock Cycle Number Reduction

To decrease the clock cycle number, a CCSA architecture which can perform one three-input carry-save addition or two serial two-input carry-save additions is proposed to substitute for the one-level CSA architecture [4]. Two cells of the one-level CSA architecture in Figure.2each cell is one conventional FA which can perform the three-input carry-save addition. Two cells of the proposed configurable FA (CFA) circuit. If α = 1, CFA is one FA and can perform one three-input carry-save addition (denoted as 1F_CSA).

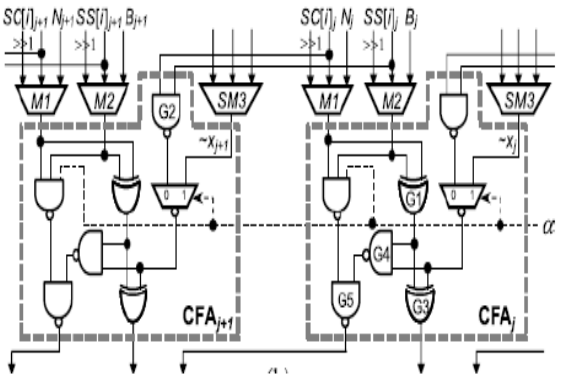

FIG.2. Carry Full Adder Circuit

Otherwise, it is two half-adders (HAs) and can perform two serial two-input carry-save additions (denoted as 2H_CSA). In this case, G1 of CF Aj and G2 of CFAj+1 will act as HA1 j and G3, G4, and G5 of CF Aj will behave as HA2j. Moreover, we modify the 4-to-1 multiplexer M3 into a simplified multiplier SM3 because one of its inputs is zero, where the INVERT operation. Note that M3 has been replaced by SM3 in the proposed one-level CCSA architecture.

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

3.PROPOSED ADVANCED ENCRYPTION STANDARD

AES is based on a design principle known as a substitution-permutation network, combination of both substitution and permutation, and is fast in both software and hardware. Its predecessor DES, AES does not use a Festal network. AES is a variant of Rijndael which has a fixed block size of 128 bits, and a key size of 128, 192, or 256 bits. By contrast, the Rijndael specification per se is specified with block and key sizes that may be any multiple of 32 bits, both with a minimum of 128 and a maximum of 256 bits [9]. AES operates on a 4×4 column-major order matrix of bytes, termed the state, although some versions of Rijndael have a larger block size and have additional columns in the state. Most AES calculations are done in a special finite field. The key size used for an AES cipher specifies the number of repetitions of transformation rounds that convert the input, called the plaintext, into the final output, called the cipher text. The numbers of cycles of repetition are as follows:

- 10 cycles of repetition for 128-bit keys.
- 12 cycles of repetition for 192-bit keys.
- 14 cycles of repetition for 256-bit keys.

Each round consists of several processing steps, each containing four similar but different stages, including one that depends on the encryption key itself [9]. A set of reverse rounds are applied to transform cipher text back into the original plaintext using the same encryption key.

3.1. High-Level Description Of The Algorithm

- Key Expansions
  - Round keys are derived from the cipher key using Rijndael's key schedule. AES requires a separate

    128-bit round key block for each round plus one more.
- Initial Round
  - Add Round Key—each byte of the state is combined with a block of the round key using bitwise xor.
- Rounds
  - Sub Bytes—a non-linear substitution step where each byte is replaced with another according to a lookup table.
  - Shift Rows—a transposition step where the last three rows of the state are shifted cyclically a certain number of steps.
  - Mix Columns—a mixing operation which operates on the columns of the state, combining the four bytes in each column.
  - Add Round Key
- Final Round (no Mix Columns)
  - Sub Bytes
  - Shift Rows
  - Add Round Key.

3.2 The Subbytes Step

In the Sub Bytes step, each byte $a_{i,j}$ in the *state* matrix is replaced with a Sub Byte $S(a_{i,j})$ using an 8-bit substitution box, the

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Rijndael S-box. This operation provides the non-linearity in the cipher. The S-box used is derived from the multiplicative inverse over GF (*28*), known to have good non-linearity properties. To avoid attacks based on simple algebraic properties, the S-box is constructed by combining the inverse function with an invertible affine transformation[4]. The S-box is also chosen to avoid any fixed points (and so is a derangement), i.e., $S(a_{i,j}) \neq a_{i,j,}$, and also any opposite fixed points, i.e., $S(a_{i,j}) \oplus a_{i,j} \neq 0\text{xFF}$. While performing the decryption, Inverse Sub Bytes step is used, this requires first taking the affine transformation and then finding the multiplicative inverse.

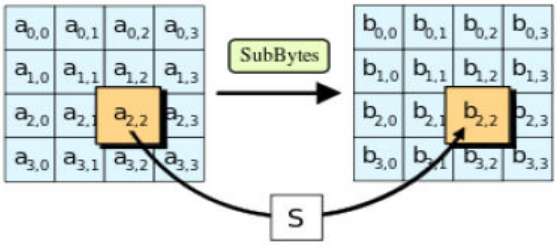

FIG.3. Diagram of Sub Bytes

3.3 THE SHIFT ROWS STEP

The Shift Rows step operates on the rows of the state; it cyclically shifts the bytes in each row by a certain offset. For AES, the first row is left unchanged. Each byte of the second row is shifted one to the left. Similarly, the third and fourth rows are shifted by offsets of two and three respectively [7]. For blocks of sizes 128 bits and 192 bits, the shifting pattern is the same. Row n is shifted left circular by n-1 bytes. In this way, each column of the output state of the Shift Rows step is composed of bytes from each column of the input state. (Rijndael variants with a larger block size have slightly different offsets).

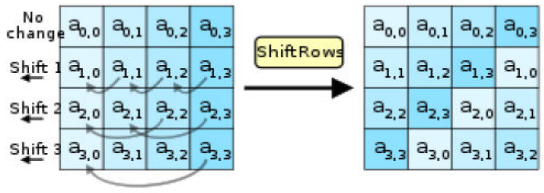

FIG.4 Diagram of Shift Rows

For a 256-bit block, the first row is unchanged and the shifting for the second, third and fourth row is 1 byte, 3 bytes and 4 bytes respectively. This change only applies for the Rijndael cipher when used with a 256-bit block, as AES does not use 256-bit blocks. The importance of this step is to avoid the columns being linearly independent, in which case, AES degenerates into four independent block cipher[10].

3.4. The Mix columns Step

In the Mix Columns step, the four bytes of each column of the state are combined using an invertible linear transformation. The Mix Columns function takes four bytes as input and outputs four bytes, where each input byte affects all four output bytes. Together with Shift Rows, Mix Columns provides diffusion in the cipher.

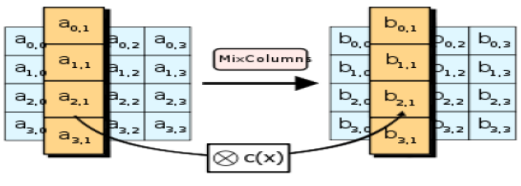

FIG.5. Diagram of Mix Columns

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Matrix multiplication is composed of multiplication and addition of the entries, and here the multiplication operation can be defined as this: multiplication by 1 means no change, multiplication by 2 means shifting to the left, and multiplication by 3 means shifting to the left and then performing XOR with the initial UN shifted value. After shifting, a conditional XOR with 0x1B should be performed if the shifted value is larger than 0xFF. (These are special cases of the usual multiplication in GF (*28*).) Addition is simply XOR. In more general sense, each column is treated as a polynomial over GF (*28*) and is then multiplied modulo x4+1 with a fixed polynomial c(x) = 0x03 · x3 + x2 + x + 0x02 [11]. The coefficients are displayed in their hexadecimal equivalent of the binary representation of bit polynomials from GF (2) [x]. The Mix Columns step can also be viewed as a multiplication by the shown particular MDS matrix in the finite field GF (*28*). This process is described further in the article Rijndael mix columns.

3.5. The Add round key Step

In the Add Round Key step, the sub key is combined with the state. For each round, a sub key is derived from the main key using Rijndael's key schedule; each sub key is the same size as the state. The sub key is added by combining each byte of the state with the corresponding byte of the sub key using bitwise XOR.

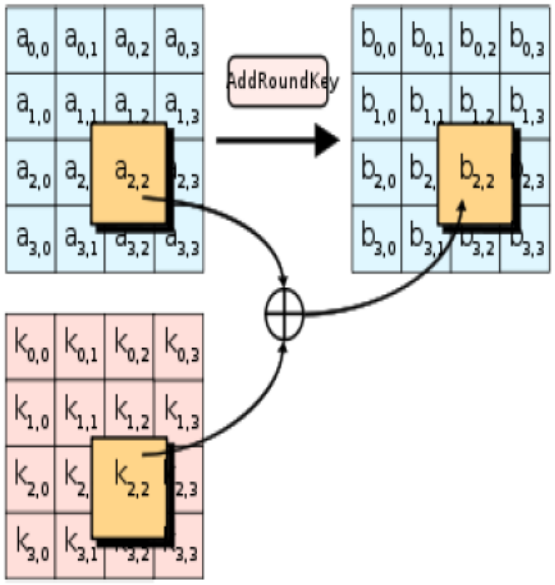

FIG.6. Diagram of Add Round Key

On systems with 32-bit or larger words, it is possible to speed up execution of this cipher by combining the Sub Bytes and Shift Rows steps with the Mix Columns step by transforming them into a sequence of table lookups. This requires four 256-entry 32-bit tables, and utilizes a total of four kilobytes (4096 bytes) of memory one kilobyte for each table. A round can then be done with 16 table lookups and 12 32-bit exclusive-or operations, followed by four 32-bit exclusive-or operations in the Add Round Key. If the resulting four-kilobyte table size is too large for a given target platform, the table lookup operation can be performed with a single 256-entry 32-bit (i.e. 1 kilobyte) table by the use of circular rotates. Using a byte-oriented approach, it is possible to combine the Sub Bytes, Shift

International Journal for Innovative Engineering and Management Research
A Peer Revieved Open Access International Journal
www.ijiemr.org

Rows, and Mix Columns steps into a single round operation [12].

5. RESULT

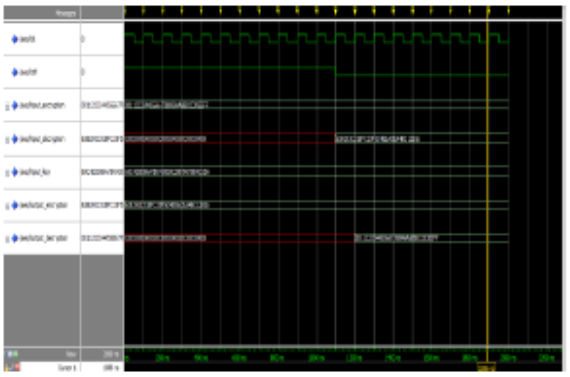

FIG.6.Output Function

6. CONCLUSION

To enhance the performance of Montgomery MM while maintaining the low hardware complexity, this paper has modified the SCS-based Montgomery multiplication algorithm a low-cost and high-performance Montgomery modular multiplier. The multiplier used one-level CCSA architecture and skipped the unnecessary carry-save addition operations to largely reduce the critical path delay and required clock cycles for completing one MM operation. FCS-based multipliers maintain the input and output operands of the Montgomery MM in the carry-save format to escape from the format conversion, leading to fewer clock cycles but larger area than SCS-based multiplier.In Future, for cryptographers, a cryptographic "break" is anything faster than a brute force performing one trial decryption for each key (see Cryptanalysis). This includes results that are infeasible with current technology. The largest successful publicly known brute force attack against any block-cipher encryption was against a 64-bit RC5 key.

ACKNOWLEDGEMENT

We are expressing our thanks to all Faculty members and Skilled Assistants of Electronics and Communication Engineering department and my Friends who helped me in every possible way. Last but not least I thank my Parents for their moral support.

REFERENCES

1. Amber.P, Pinckney.N, and Harris, D. M. "Parallel high-radix Montgomery multipliers,"(2008) in Proc. 42nd Asilomar Conf. Signals, Syst., Comput., pp. 772–776.
2. Bunimov.V, Schimmler.M, and Tolg.B, "A complexity-effective version of Montgomery's algorihm," (2002) in Proc. Workshop Complex.Effective Designs.
3. Gang.F, "Design of modular multiplier based on improved Montgomery algorithm and systolic array," (2006) in Proc. 1st Int. Multi-Symp. Comput. Co mput. Sci., vol. 2. Jun. 2006, pp. 356–359.
4. Han, J. Wang S., Huang W., Yu Z., and Zeng X, "Parallelization of radix-2 Montgomery multiplication on multicore platform,"(2013) IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 12, pp. 2325–2330,.
5. Kuang S.-R., Wang J.-P., Chan K.-C., and Hsu. H.-W., "Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems," (2013) IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 11,pp. 1999–2009,.

6. McIvor.C, McLoone.M, and McCanny, J. V. "Modified Montgomery modular multiplication and RSA exponentiation techniques,"(2004) IEE Proc.-Comput. Digit. Techn., vol. 151, no. 6, pp. 402–408,.
7. Miyamoto A., Homma N., Aoki, T. and Satoh.A, "Systematic design of RSA processors based on high-radix Montgomery multipliers,"(2011) IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 7, pp. 1136–1146.
8. Neto, J. C. Tenca A. F., and Ruggiero W. V., "A parallel k-partition method to perform Montgomery multiplication,"(2011) in Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit., Processors, , pp. 251–254.
9. Sassaw.G,. Jimenez.C.J, and Valencia.M, "High radix implementation of Montgomery multipliers with CSA," (2010) in Proc. Int. Conf. Micro electron., Dec. 2010, pp. 315–318.
10. Saemen.J and Rijmen.V, The block cipher Rijndael, Smart Card research and Applications, (2010)LNCS 1820, Springer-Verlag, pp. 288-296
11. Wang S.-H., Lin W.-C "Fast scalable radix-4 Montgomery modular multiplier," (2012) in Proc. IEEE Int. Symp. Circuits Syst., , pp. 3049–3052.
12. Yee.A, Guideline for Implementing Cryptography in the Federal Government, National Institute of Standards and Technology, (1999),NIST Special Publication 800-21.