International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 17
November 2016

Low Cost And High Performance Of Vlsi Architecture For Reconfigurable Montgomery Modular Multiplication

Mandalaneni Jaya[1], Shaik Masthan Sharif[2]

[1]PG Scholar, Dept of ECE, EVM College Of Engineering &Technology, Narsaraopet,Guntur Dist,AP,India.

[2]Assistant Professor, Dept of ECE, EVM College Of Engineering &Technology, Narsaraopet, Guntur Dist, AP,India.

ABSTRACT:

This paper proposes a simple and efficient Montgomery multiplication algorithm such that the low-cost and high-performance Montgomery modular multiplier can be implemented accordingly. The proposed 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. In addition, a 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. As a result, the extra clock cycles for operand pre computation and format conversion can be hidden and high throughput can be obtained. Experimental results show that the proposed Montgomery modular multiplier can achieve higher performance and significant area–time product improvement when compared with previous designs.

Index Terms— Carry-save addition, low-cost architecture, Montgomery modular multiplier, public-key cryptosystem.

INTRODUCTION

I N 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 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 algorithm as shown in step 4 of Fig. 1 requires long carry propagation for large operands in binary representation. To solve this problem, several approaches

Carry Save Adders and Redundant Representation :

The core operation of most algorithms for modular multiplication is addition. There are several different methods for addition in hardware: carry ripple addition, carry select addition, carry look ahead addition and others [8]. The disadvantage of these methods is the carry propagation, which is directly proportional to the length of the operands. This is not a big problem for operands of size 32 or 64 bits but the typical operand size in cryptographic applications range from 160 to 2048 bits. The resulting delay has a significant influence on the time complexity of these adders. The carry save adder seems to be the most cost effective adder for our application. Carry save addition is a method for an addition without carry propagation. It is simply a parallel ensemble of n full-adders without any horizontal connection. Its function is to add three n-bit integers X, Y, and Z to produce two integers C and S as results such that C + S = X + Y + Z, where C represents the carry and S the sum.

When carry save adders are used in an algorithm one uses a notation of the form (S, C) = X + Y + Z to indicate that two results are produced by the addition. The results are now represented in two binary words, an n-bit word S and an (n+1) bit word C. Of course, this representation is redundant in the sense that we can represent one value in several different ways. This redundant representation has the advantage that the arithmetic operations are fast, because there is no carry propagation. On the other hand, it brings to the fore

one basic disadvantage of the carry save adder: • It does not solve our problem of adding two integers to produce a single result. Rather, it adds three integers and produces two such that the sum of these two is equal to that of the three inputs. This method may not be suitable for applications which only require the normal addition.

Montgomery Multiplication Algorithm:

The Montgomery algorithm [1, Algorithm 1a] computes P = (X*Y* (2 n) -1) mod M. The idea of Montgomery [2] is to keep the lengths of the intermediate results

smaller than n+1 bits. This is achieved by interleaving the computations and additions of new partial products with divisions by 2; each of them reduces the bit length of the intermediate result by one. For a detailed treatment of the Montgomery algorithm, the reader is referred to [2] and [1]. The key concepts of the Montgomery algorithm [1, Algorithm 1b] are the following: • Adding a multiple of M to the intermediate result does not change the value of the final result; because the result is computed modulo M. M is an odd number. • After each addition in the inner loop the least significant bit (LSB) of the intermediate result is inspected. If it is 1, i.e., the intermediate result is odd, we add M to make it even. This even number can be divided by 2 without remainder. This division by 2 reduces the intermediate result to n+1 bits again. • After n steps these divisions add up to one division by 2 n . The Montgomery algorithm is very easy to implement since it operates least significant bit first and does not require any comparisons.

Algorithm MM:
Radix-2 Montgomery modular multiplication

Inputs : A, B, N (modulus)
Output : $S[k]$

1. $S[0] = 0$;
2. for $i = 0$ to $k - 1$ {
3. $q_i = (S[i]_0 + A_i \times B_0) \bmod 2$;
4. $S[i+1] = (S[i] + A_i \times B + q_i \times N) / 2$;
5. }
6. if $(S[k] \geq N)\ S[k] = S[k] - N$;
7. return $S[k]$;

Fig. 1. MM algorithm.

Montgomery Multiplication :

Algorithm SCS-based MM:
SCS-based Montgomery multiplication

Inputs : A, B, N (modulus)
Outputs : $S[k+2]$

1. $SS[0] = 0$; $SC[0] = 0$;
2. for $i = 0$ to $k + 1$ {
3. $q_i = (SS[i]_0 + SC[i]_0 + A_i \times B_0) \bmod 2$;
4. $(SS[i+1], SC[i+1]) = (SS[i]+SC[i] + A_i \times B + q_i \times N) / 2$;
5. }
6. $S[k+2] = SS[k+2] + SC[k+2]$;
7. return $S[k+2]$;

Fig. 2. SCS-based Montgomery multiplication algorithm.

Fig. 1 shows the radix-2 version of the Montgomery MM algorithm (denoted as MM algorithm). As mentioned earlier, the Montgomery modular product S of A and B can be obtained as $S = A \times B \times R^{-1} \pmod N$, where R^{-1} is the inverse of R modulo N. That is, $R \times R^{-1} = 1 \pmod N$. Note that, the notation Xi in Fig. 1 shows the ith bit of X in binary representation. In addition, the notation Xi: j indicates a segment of X from the ith bit to jth bit. Since the convergence range of S in MM algorithm is $0 \le S < 2N$, an additional operation $S = S - N$ is required

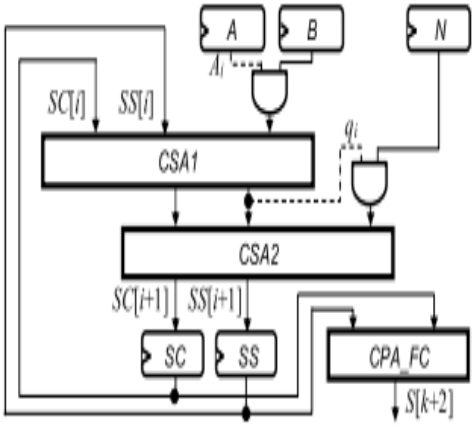


Fig. 3. SCS-MM-1 multiplier.

to remove the oversize residue if $S \ge N$. To eliminate the final comparison and subtraction in step 6 of Fig. 1, Walter [22] changed the number of iterations and the value of R to k + 2 and 2k+2 mod N, respectively. Nevertheless, the long carry propagation for the very large operand addition still restricts the performance of MM algorithm.

SCS-Based Montgomery Multiplication :

To avoid the long carry propagation, the intermediate result S of shifting modular addition can be kept in the carry-save representation (SS, SC), as shown in Fig. 2. Note that the number of iterations in Fig. 2 has been changed from k to k + 2 to remove the final comparison and subtraction [22]. However, the format conversion from the carry-save format of the final modular product into its binary format is needed, as shown in step 6 of Fig. 2. Fig. 3 shows the architecture of SCS-based MM algorithm proposed in [5] (denoted as SCS-MM-1 multiplier) composed of one two-level CSA architecture and one format converter, where the dashed line denotes a 1-bit signal. In [5], a 32-bit CPA with multiplexers and registers (denoted as CPA_FC), which adds two 32-bit inputs and generates a 32-bit output at every clock cycle, was adopted for the format conversion. Therefore, the 32-bit CPA_FC will take 32 clock cycles to complete the format conversion of a 1024-bit SCS-based Montgomery multiplication.

The extra CPA_FC probably enlarges the area and the critical path of the SCS-MM-1 multiplier. The works in [6] and [7] pre computed D = B + N so that the computation of Ai × B + qi × N in step 4 of Fig. 2 can be simplified into one selection operation. One of the

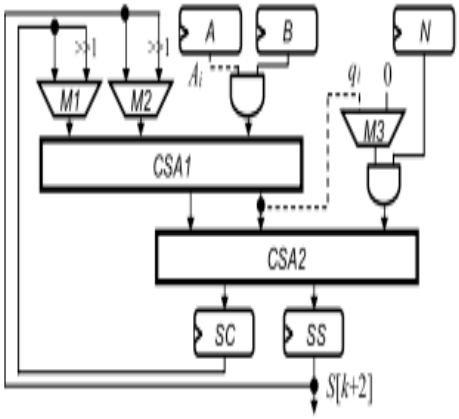


Fig. 4. SCS-MM-2 multiplier.

operands 0, N, B, and D will be chosen if (Ai , qi) = (0, 0), (0, 1), (1, 0), and (1, 1), respectively. As a result, only one-level CSA architecture is required in this multiplier to perform the carry-save addition at the expense of one extra 4-to-1 multiplexer and one additional register to store the operand D. However, they did not present an effective approach to remove the CPA_FC for format conversion and thus this kind of multiplier still suffers from the critical path of CPA_FC. On the other hand, Zhang et al. [8] reused the two-level CSA architecture to perform the format conversion so that the CPA_FC can be removed. That is, S[k + 2] = SS[k + 2] + SC[k + 2] in step 6 of Fig. 2 is replaced with the repeated carry-save addition operation (SS[k + 2], SC[k + 2]) = SS[k + 2] + SC[k + 2] until SC[k + 2] = 0. Fig. 4 shows the architecture of the Montgomery multiplier proposed in [8] (denoted as SCS-MM-2 multiplier). Note that the select signals of multiplexers M1 and M2 in Fig. 4 generated by the control part are not shown in Fig. 4 for the sake of simplicity. However, the extra clock cycles for format conversion are dependent on the longest carry propagation chain in SS[k+2]+SC[k+2] and about k/2 clock cycles are required in the worst case because two-level CSA architecture is adopted in [8]

FCS-Based Montgomery Multiplication:

To avoid the format conversion, FCS-based Montgomery multiplication maintains A, B, and S in the carry save representations (AS, AC), (BS, BC), and (SS, SC), respectively. McIvor et al. [9] proposed two FCS based Montgomery multipliers, denoted as FCS-MM-1 and FCS-MM-2 multipliers, composed of one five-totwo (three-level) and one four-to-

two (two-level) CSA architecture, respectively. The algorithm and architecture of the FCS-MM-1 multiplier are shown in Figs. 5 and 6, respectively. The barrel register full adder (BRFA)

Algorithm FCS-MM-1:
FCS-based Montgomery multiplicati
Inputs : *AS*, *AC*, *BS*, *BC*, *N* (modulu
Outputs : *SS*[*k*+2], *SC*[*k*+2]
1. *SS*[0] = 0; *SC*[0] = 0;
2. for *i* = 0 to *k* + 1 {
3. $q_i = (SS[i]_0 + SC[i]_0 + A_i \times (BS_0$
4. (*SS*[*i*+1], *SC*[*i*+1]) = (*SS*[*i*] + *SC*
+
5. }
6. return *SS*[*k*+2], *SC*[*k*+2];

Fig. 5. FCS-MM-1 Montgomery multiplication algorithm.

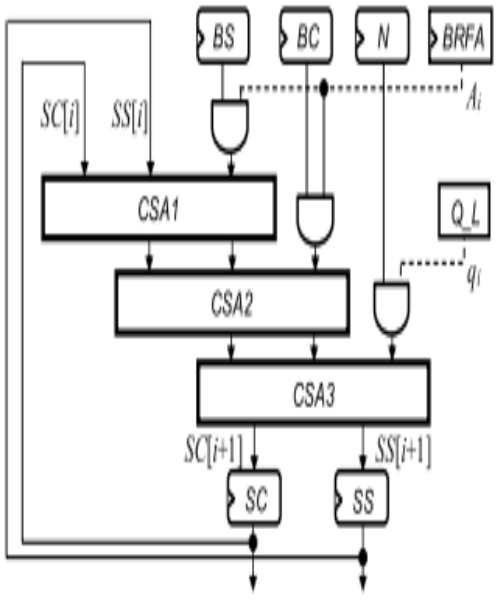


Fig. 6. FCS-MM-1 multiplier.

in Fig. 6 consists of two shift registers for storing AS and AC, a full adder (FA), and a flip-flop (FF). For more details about BRFA, please refer to [9] and [10]. On the other hand, the FCS-MM-2 multiplier proposed in [9] adds up BS, BC, and N into DS and DC at the beginning of each MM. Therefore, the depth of the CSA tree can be reduced from three to two levels. Nevertheless, the FCS-MM-2 multiplier needs two extra 4-to-1 multiplexers addressed by Ai and qi and two more registers to store DS and DC to reduce one level of CSA tree. Therefore, the critical path of the FCS-MM-2 multiplier may be slightly reduced with a significant increase in hardware area when compared with the FCS-MM-1 multiplier. Table I summarizes and roughly compares the area complexity and critical path delay of the above-mentioned radix-2 Montgomery multipliers according to the normalized area and delay listed in Table II with respect to the TSMC 90-nm cell library information. In Table I, the

Multiplier	FC	Area Complexity	Area Ratio	Critical Path Delay	Delay Ratio
SCSA-MM-1 [9]	Yes	$2k\times d_{FA}+4k\times d_{MUX}+k\times d_{FF}+A_{FA\_FC}$	$6.76k\times d_{FA}$*	$\max(2\times T_{AND}+2\times T_{FA}, (CPA\_FC))$	$2.08\times T_{FA}$*
SCSA-MM-2 [9]	Yes	$2k\times d_{FA}+4k\times d_{MUX}+k\times d_{FF}+2k\times d_{AND}+3k\times d_{MUX}$	$8.24k\times d_{FA}$	$2\times T_{MUX}+T_{AND}+2\times T_{FA}$	$3.24\times T_{FA}$
FCS-MM-1 [9]	No	$3k\times d_{FA}+5k\times d_{MUX}+2k\times d_{FF}+3k\times d_{AND}$	$10.48k\times d_{FA}$	$2\times T_{MUX}+T_{AND}+3\times T_{FA}$	$4.02\times T_{FA}$
FCS-MM-2 [9]	No	$2k\times d_{FA}+3k\times d_{MUX}+2k\times d_{FF}+2k\times d_{MUX}$	$12.56k\times d_{FA}$	$2\times T_{MUX}+T_{AND}+T_{MUX}+2\times T_{FA}$	$3.75\times T_{FA}$

* the ratio does not consider the extra CPA_FC

TABLE I ANALYSIS OF AREA AND DELAY OF DIFFERENT DESIGNS

notations AG and TG denote the area and delay of a cell G, respectively, and τ () denotes the critical path delay of circuit . Note that ASR in Table I denotes the area of a shift register, and we assume that ASR is approximate to the sum of AREG and AMUX2. In addition, the area and delay ratios of the SCS-MM-1 multiplier in Table I do not take that of CPA_FC into consideration because they are signifi- cantly dependent on the design of CPA_FC. Generally speaking, SCS-based multipliers have lower area complexity than FCS-based Montgomery multipliers. However, extra clock cycles for format conversion possibly lower the performance of SCS-based multipliers. To further enhance the performance of the SCS-based multiplier, both the critical path delay and clock cycles for completing one multiplication must be reduced while maintaining the low hardware complexity.

PROPOSED MONTGOMERY MULTIPLICATION :

In this section, we 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.

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, we can precompute D = B + N and reuse the one-level CSA architecture to perform B+N and the format conversion. Fig. 7(a) and (b) shows the modified SCS-based Montgomery multiplication (MSCS-MM) algorithm and one possible hardware architecture, respectively. The Zero_D circuit in Fig. 7(b) 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 according to step 7 of Fig. 7(a). 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 precompute Ai and qi in iteration i−1

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 17
November 2016

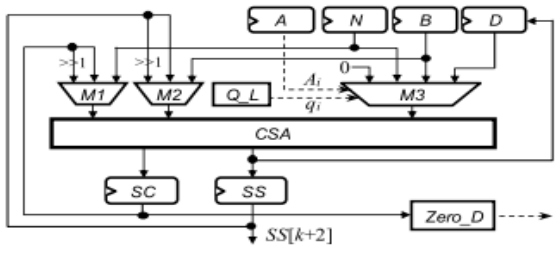
(this will be explained more clearly in Section III-C) so that they can be used to immediately select the desired input operand from 0, N, B, and D through the multiplexer M3 in iteration i. Therefore, the critical path delay of the MSCS-MM multiplier can be reduced into TMUX4 + TFA. However, in addition to performing the

Algorithm Modified SCS-MM:
Modified SCS-based Montgomery multiplication

Inputs : A, B, N (modulus)
Output : $SS[k+2]$

1. $(SS, SC) = (B + N + 0)$;
2. while $(SC \mathrel{!}= 0)$
3. $\quad (SS, SC) = (SS + SC + 0)$;
4. $D = SS$;
5. $SS[0] = 0$; $SC[0] = 0$;
6. for $i = 0$ to $k + 1$ {
7. $\quad q_i = (SS[i]_0 + SC[i]_0 + A_i \times B_0) \bmod 2$;
8. $\quad$ if $(A_i = 0$ and $q_i = 0)$ $x = 0$;
9. $\quad$ if $(A_i = 0$ and $q_i = 1)$ $x = N$;
10. $\quad$ if $(A_i = 1$ and $q_i = 0)$ $x = B$;
11. $\quad$ if $(A_i = 1$ and $q_i = 1)$ $x = D$;
12. $\quad (SS[i+1], SC[i+1]) = (SS[i] + SC[i] + x) / 2$;
13. }
14. while $(SC[k+2] \mathrel{!}= 0)$
15. $\quad (SS[k+2], SC[k+2]) = (SS[k+2] + SC[k+2] + 0)$;
16. return $SS[k+2]$;

(a)



(b)

Fig. 7. (a) Modified SCS-based Montgomery multiplication algorithm. (b) MSCS-MM multiplier.

three-input carry-save additions [i.e., step 12 of Fig. 7(a)] 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…1112 and SC = 000…0012, the one-level CSA architecture needs k clock cycles to complete SS + SC.

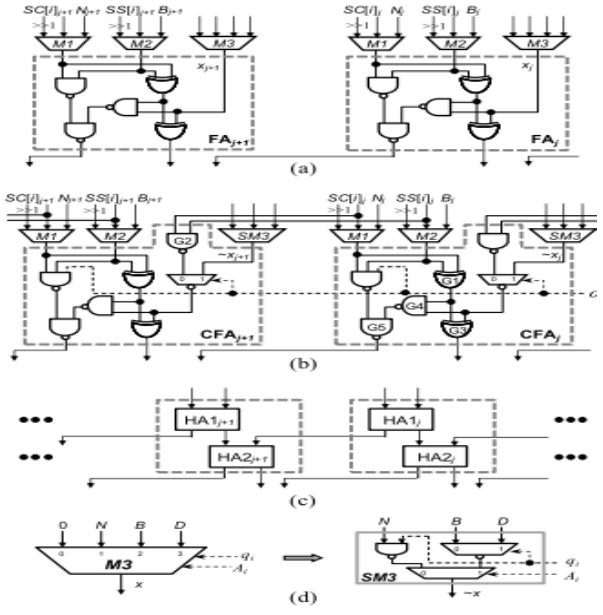


Fig. 8. (a) Conventional FA circuit. (b) Proposed CFA circuit. (c) Two serial HAs. (d) Simplified multiplexer SM3.

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.

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 in Fig. 7(b). Fig. 8(a) shows two cells of the one-level CSA architecture in Fig. 7(b), each cell is one conventional FA which can perform the three-input carry-save addition. Fig. 8(b) shows two cells of the proposed configurable FA (CFA) circuit. If $\alpha = 1$, CFA is one FA and can perform one three-input carry-save addition (denoted as 1F_CSA). Otherwise, it is two half-adders (HAs) and can perform two serial two-input carry-save additions (denoted as 2H_CSA), as shown in Fig. 8(c). In this case, G1 of CFAj and G2 of CFAj+1 in Fig. 8(b) will act as HA1 j in Fig. 8(c), and G3, G4, and G5 of CFAj in Fig. 8(b) will behave as HA2 j in Fig. 8(c). Moreover, we modify the 4-to-1 multiplexer M3 in Fig. 7(b) into a simplified multiplier SM3 as shown in Fig. 8(d) because one of its inputs is zero, where ~ denotes the INVERT operation. Note that M3 has been replaced

$$\begin{array}{cccccc} SS[i]_{k+1} & SS[i]_k & \cdots & SS[i]_2 & SS[i]_1 & SS[i]_0 \\ SC[i]_{k+1} & SC[i]_k & \cdots & SC[i]_2 & SC[i]_1 & SC[i]_0 \\ x_{k+1} & x_k & \cdots & x_2 & x_1 & x_0 \\ \hline SS[i+1]_k & SS[i+1]_{k-1} & \cdots & SS[i+1]_1 & SS[i+1]_0 & 0 \\ SC[i+1]_{k+1} & SC[i+1]_k & SC[i+1]_{k-1} & \cdots & SC[i+1]_1 & SC[i+1]_0 \end{array}$$

Fig. 9. Three-to-two carry-save addition at the ith iteration of Fig. 7.

by SM3 in the proposed one-level CCSA architecture shown in Fig. 8(b). According to the delay ratio shown in Table II, TS M3 (i.e., 0.68 × TFA) is approximate to TMUX3 (i.e., 0.63 × TFA) and TMUXI2 (i.e., 0.23 × TFA) is smaller than TXOR2 (i.e., 0.34×TFA). Therefore, the critical path delay of the proposed one-level CCSA architecture in Fig. 8(b) is approximate to that of the one-level CSA architecture in Fig. 8(a). As a result, steps 3 and 15 of Fig. 7(a) can be replaced with (SS, SC) = 2H_CSA(SS, SC) and (SS[k + 2], SC[k + 2]) = 2H_CSA (SS[k + 2], SC[k + 2]) to reduce the required clock cycles by approximately a factor of two while maintaining a short critical path delay. In addition, we also skip the unnecessary

operations in the for loop (steps 6 to 13) of Fig. 7(a) to further decrease the clock cycles for completing one Montgomery MM. The crucial computation in the for loop of Fig. 7(a) is performing the following three-to-two carry-save addition: (SS[i + 1], SC[i + 1]) = (SS[i] + SC[i] + x)/2 (1) where the variable x may be 0, N, B, or D depending on the values of Ai and qi . The computation process of (1) is shown in Fig. 9. When Ai = 0 and qi = 0, x is equal to 0 and SS[i]0 must be equal to SC[i]0 because the sum of SS[i]0 + SC[i]0 + x0 is equal to 0. That is, if Ai = 0 and qi = 0, then SS[i]0 = SC[i]0. Based on this observation, we can conclude that the sum of the carry propagation addition SS[i +1]k+1:0 + SC[i + 1]k+1:0 is equal to the sum of the carry propagation addition SS[i]k+1:1 + SC[i]k+1:1 when Ai = qi = 0 and SS[i]0 = SC[i]0 = 0. As a result, the computation of (1) in iteration i can be skipped if we directly right shift the outputs of one-level CSA architecture in the (i − 1)th iteration by two bit positions (i.e., divided by 4) instead of one bit position (i.e., divided by 2) when Ai = qi = 0 and SS[i]0 = SC[i]0 = 0. Accordingly, the signal skipi+1 used in the ith iteration to indicate whether the carry-save addition in the (i + 1)th iteration will be skipped can be expressed as skipi+1 = ∼(Ai+1 ∨ qi+1 ∨ SS[i + 1]0) (2) where ∨ represents the OR operation. If skipi+1 generated in the ith iteration is 0, the carry-save addition of the (i + 1)th iteration will not be skipped. In this case, qi+1 and Ai+1 produced in the ith iteration can be stored in FFs and then used to fast select the value of x in the (i +1)th iteration. Otherwise (i.e., skipi+1 = 1), SS[i + 1] and SC[i + 1] produced in the ith iteration must be right shifted by two bit positions and the next clock cycle will go to iteration i + 2 to skip the carry-save addition of the (i + 1)th iteration. In this situation, not only qi+1 and Ai+1 but also qi+2 and Ai+2 must be produced and stored to FFs in the ith iteration to immediately select the value of x in the (i + 2)th iteration without lengthening the critical path. Therefore, the selection signals (denoted as qˆ and Aˆ) for choosing the proper value of x in the next clock cycle must be picked from (qi+1, Ai+1) or (qi+2, Ai+2) according to the skipi+1 signal produced in the ith iteration. That is, (qˆ, Aˆ) = (qi+2, Ai+2) if skipi+1 = 1. Otherwise, (qˆ, Aˆ) = (qi+1, Ai+1)

International Journal of Research
Available at https://edupediapublications.org/journals
p-ISSN: 2348-6848
e-ISSN: 2348-795X
Volume 03 Issue 17
November 2016

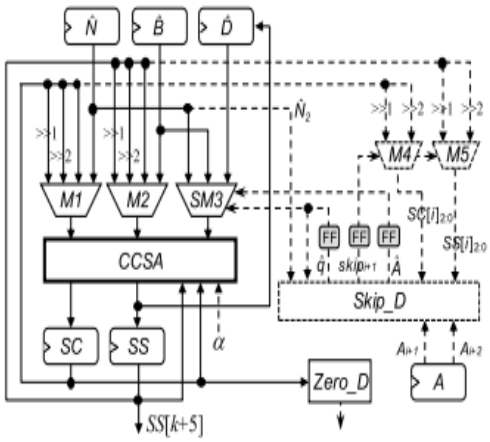


Fig. 11. SCS-MM-New multiplier.

multiplier SM3, one skip detector Skip_D, one zero detector Zero_D, and six registers. Skip_D is developed to generate skipi+1, qˆ, and Aˆ in the ith iteration. Both M4 and M5 in Fig. 11 are 3-bit 2-to-1 multiplexers and they are much smaller than k-bit multiplexers M1, M2, and SM3. In addition, the area of Skip_D is negligible when compared with that of the k-bit one-level CCSA architecture. Similar to Fig. 4, the select signals of multiplexers M1 and M2 in Fig. 11 are generated by the control part, which are not depicted for the sake of simplicity.

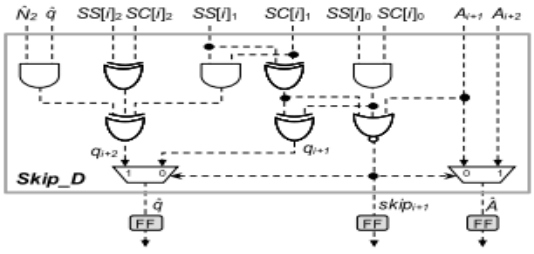


Fig. 12. Skip detector Skip_D.

At the beginning of Montgomery multiplication, the FFs stored skipi+1, qˆ, Aˆ are first reset to 0 as shown in step 1 of SCS-MM-New algorithm so that Dˆ = Bˆ +Nˆ can be computed via the one-level CCSA architecture. When performing the while loop, the skip detector Skip_D shown in Fig. 12 is used to produce skipi+1, qˆ, and Aˆ. The Skip_D is composed of four XOR gates, three AND gates, one NOR gate, and two 2-to-1 multiplexers. It first generates the qi+1, qi+2, and skipi+1 signal in the ith iteration according to (5), (7), and (8), respectively, and then selects the correct qˆ and Aˆ according to skipi+1. At the end of the ith iteration, qˆ, Aˆ, and skipi+1 must be stored to FFs. In the next clock cycle of the ith iteration, SM3 outputs a proper x according to qˆ and Aˆ generated in the ith iteration as shown in steps 8–11, and M1 and M2 output the correct SC and SS according to skipi+1 generated in the ith iteration. If skipi+1 = 0, SC 1 and SS 1 are selected. Otherwise, SC 2 and SS 2 are selected. That is, the right-shift 1-bit operations in steps 12 and 15 of SCS-MM-New algorithm are performed together in the next clock cycle of iteration i. In addition, M4 and M5 also select and output the correct SC[i]2:0 and SS[i]2:0

according to skipi+1 generated in the ith iteration.

Applications:

1. Digital Signal Processing
2. CSA architectures…etc..

Advantages:

1. Speed
2. Cost, delay

CONCLUSION :

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. To enhance the performance of Montgomery MM while maintaining the low hardware complexity, this paper has modified the SCS-based Montgomery multiplication algorithm and proposed a low-cost and high-performance Montgomery modular multiplier. The proposed 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. Experimental results showed that the proposed approaches are indeed capable of enhancing the performance of radix-2 CSA-based Montgomery multiplier while maintaining low hardware complexity.

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, Feb. 1978.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in Advances in Cryptology. Berlin, Germany: Springer-Verlag, 1986, pp. 417–426.

[3] N. Koblitz, "Elliptic curve cryptosystems," Math. Comput., vol. 48, no. 177, pp. 203–209, 1987.

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

[5] Y. S. Kim, W. S. Kang, and J. R. Choi, "Asynchronous implementation of 1024-bit modular processor for RSA cryptosystem," in Proc. 2nd IEEE Asia-Pacific Conf. ASIC, Aug. 2000, pp. 187–190.

[6] V. Bunimov, M. Schimmler, and B. Tolg, "A complexity-effective version of Montgomery's algorihm," in Proc. Workshop Complex. Effective Designs, May 2002.

[7] H. Zhengbing, R. M. Al Shboul, and V. P. Shirochin, "An efficient architecture of 1024-bits cryptoprocessor for RSA cryptosystem based on modified Montgomery's algorithm," in Proc. 4th IEEE Int. Workshop Intell. Data Acquisition Adv. Comput. Syst., Sep. 2007, pp. 643–646.

[8] Y.-Y. Zhang, Z. Li, L. Yang, and S.-W. Zhang, "An efficient CSA architecture for Montgomery modular multiplication," Microprocessors Microsyst., vol. 31, no. 7, pp. 456–459, Nov. 2007.

[9] C. McIvor, M. McLoone, and J. V. McCanny, "Modified Montgomery modular multiplication and RSA exponentiation techniques," IEE Proc.- Comput. Digit. Techn., vol. 151, no. 6, pp. 402–408, Nov. 2004.

[10] S.-R. Kuang, J.-P. Wang, K.-C. Chang, and H.-W. Hsu, "Energy-efficient high-throughput Montgomery modular multipliers for RSA cryptosystems," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 11, pp. 1999–2009, Nov. 2013.

pp. 1136–1146, Jul. 2011.

[11] J. C. Neto, A. F. Tenca, and W. V. Ruggiero, "A parallel k-partition method to perform Montgomery multiplication," in Proc. IEEE Int. Conf. Appl.-Specific Syst., Archit., Processors, Sep. 2011, pp. 251–254.
[12] J. Han, S. Wang, W. Huang, Z. Yu, and X. Zeng, "Parallelization of radix-2 Montgomery multiplication on multicore platform," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 21, no. 12, pp. 2325–2330, Dec. 2013.

[13] P. Amberg, N. Pinckney, and D. M. Harris, "Parallel high-radix Montgomery multipliers," in Proc. 42nd Asilomar Conf. Signals, Syst., Comput., Oct. 2008, pp. 772–776.

[14] G. Sassaw, C. J. Jimenez, and M. Valencia, "High radix implementation of Montgomery multipliers with CSA," in Proc. Int. Conf. Microelectron., Dec. 2010, pp. 315–318.

[15] A. Miyamoto, N. Homma, T. Aoki, and A. Satoh, "Systematic design of RSA processors based on high-radix Montgomery multipliers," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 19, no. 7,