VLSI Implementation of High Performance Montgomery Modular Multiplication for Crypto Graphical Application

Baana Lakshmi Narayanamma[1], B.Aruna[2]
M.Tech. PG Scholar, Gouthami Institute of Technology & Management for Women, Proddatur
Assistant Professor, Gouthami Institute of Technology & Management for Women, Proddatur

***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. 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 design.**

I. INTRODUCTION

The 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 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 $R = 2^k \bmod N$. 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 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.

Kuang et al. [10] have proposed an energy-efficient FCS-based multiplier (denoted as FCS-MMM42 multiplier) in which the superfluous operations of the four-to-two (two-level) CSA architecture are suppressed to reduce the energy dissipation and enhance the throughput. However, the FCS-MMM42

multiplier still suffers from the high area complexity and long critical path delay. Other techniques, such as parallelization, high-radix algorithm, and systolic array design [11]–[19], can be combined with the CSA architecture to further enhance the performance of Montgomery multipliers. However, these techniques probably cause a large increase in hardware complexity and power/energy dissipation [20], [21], which is undesirable for portable systems with constrained resources.

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 $S[k]$;

Fig2: *Montgomery modular multiplication*

Accordingly, this paper aims at enhancing the performance of CSA-based Montgomery multiplier while maintaining low hardware complexity. Instead of the FCS-based multiplier with two-level CSA architecture in [10], a new SCS-based Montgomery MM algorithm and its corresponding hardware architecture with only one-level CSA are proposed in this paper. The proposed algorithm and hardware architecture have the following several advantages and novel contributions over previous designs. First, the one-level CSA is utilized to perform not only the addition operations in the iteration loop of Montgomery's algorithm but also B + N and the format conversion, leading to a very short critical path and lower hardware cost. However, a lot of extra clock cycles are required to carry out B + N and the format conversion via the one-level CSA architecture. Therefore, the benefit of short critical path will be lessened.

To overcome the weakness, we then modify the one-level CSA architecture to be able to perform one three-input carry-save addition or two serial two-input carry-save additions, so that the extra clock cycles for B + N and the format conversion can be reduced by half. Finally, the condition and detection circuit, which are different with that of FCS-MMM42 multiplier in [10], are developed to pre compute quotients and skip the unnecessary carry-save addition operations in the one-level configurable CSA (CCSA) architecture while keeping a short critical path delay. Therefore, the required clock cycles for completing one MM operation can be significantly reduced. As a result, the proposed Montgomery multiplier can obtain higher throughput and much smaller area-time product (ATP) than previous Montgomery multipliers.

A. *Modular Multiplication Algorithms*

a. *Montgomery Multiplication*

Fig.1shows 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 × B × R−1 (mod N), where R−1 is the inverse of R modulo N. That is, R × R−1 = 1 (mod 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 $0 \leq S < 2N$, an additional operation S = S − N is required to remove the oversize residue if $S \geq N$. 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.

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 $S[k+2]$;

Fig2: *SCS-based Montgomery multiplication algorithm*

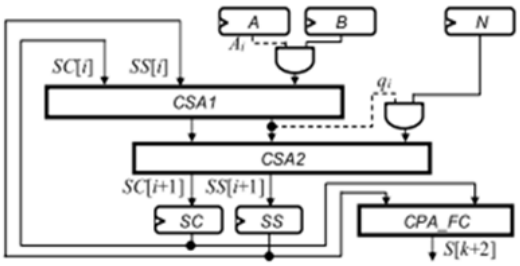


Fig 3: *SCS-MM-1 multiplier*

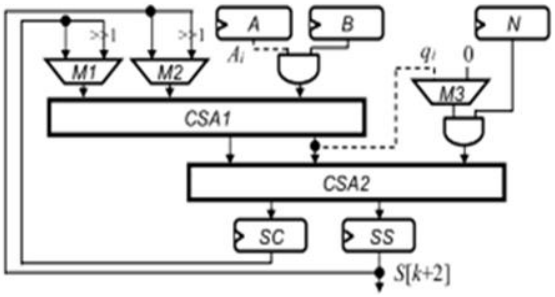


Fig 4: *SCS-MM-2 multiplier*

B .*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 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.2is 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. 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].

II. LITERATURE SURVEY

An encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption key. This has two important consequences: 1. Couriers or other secure means are not needed to transmit keys, since a message can be enciphered using an encryption key publicly revealed by the intended recipient. Only he can decipher the message, since only he knows the corresponding decryption key. 2. A message can be "signed" using a privately held decryption key. Anyone can verify this signature using the corresponding publicly revealed encryption key. Signatures cannot be forged, and a signer cannot later deny the validity of his signature.

This has obvious applications in "electronic mail" and "electronic funds transfer" systems. A message is encrypted by representing it as a number M, raising M to a publicly specified power e, and then taking the remainder when the result is divided by the publicly specified product, n, of two large secret prime numbers p and q. Decryption is similar; only a different, secret, power d is used, where e • d ≡ 1 (mod (p − 1) • (q − 1)). The security of the system rests in part on the difficulty of factoring the published divisor, n. Key Words and Phrases: digital signatures, public-key cryptosystems, privacy, authentication, security, factorization, prime number, electronic mail, message-passing, electronic funds transfer, and cryptography.

Some algorithms [1], [2], [4], [5] require extensive modular arithmetic. We propose a representation of residue classes so as to speed modular multiplication without affecting the modular addition and subtraction algorithms. Other recent algorithms for modular arithmetic appear in [3], [6]. Fix N > 1.

Define an A'-residue to be a residue class modulo N. Select a radix R co prime to N (possibly the machine word size or a power thereof) such that R > N and such that computations modulo R are inexpensive to process. Let R~l and N' be integers satisfying 0 < R'x < N and 0 < N' < R and RRX - NN' = 1. For 0 < i < N, let /' represent the residue class containing iR~x mod N.This is a complete residue system. The rationale behind this selection is our ability to quickly compute TRl mod N from T if 0 < T < RN, as shown in Algorithm REDC: function REDC(r) m «- iT mod R)N' mod R [so 0 < m < R] t <-(T+ mN)/R if t > N then return t - N else return . To validate REDC, observe mN = TN'N = -Tmod R, so t is an integer. Also, tR = Tmod N so t = TR'X mod N. Thirdly, 0 < T + mN < RN + RN, so 0 < t < 2N. If R and N are large, then T + mN may exceed the largest double-precision value.

One can circumvent this by adjusting m so -R < m < 0. Given two numbers x and y between 0 and N - 1 inclusive, let z = REDC(xy). Then z = (xy)R~x mod N, so (xR-l)(yR~x) = zRx mod N. Also, 0 < z < N, so z is the product of x and y in this representation. Other algorithms for operating on N-residues in this representation can be derived from the algorithms normally used. The addition algorithm is unchanged, since xR~x + yR~x = zR~x mod N if and only if x + y = z mod N. Also unchanged are the algorithms for subtraction, negation, equality/inequality test, multiplication by an integer, and greatest common divisor with N.

To convert an integer x to an ^-residue, compute xR mod N. Equivalently, compute REDC ((xmod N) (R2mod N)). Constants and inputs should be converted once, at the start of an algorithm. To convert an ^-residue to an integer, pad it with leading zeros and apply Algorithm REDC (thereby multiplying it by R'1 mod N). To invert an TV-residue, observe (xR~x) ~l = zR'1 mod N if and only if z = R2x~l mod N. For modular division, observe (xR~l) (yR~x)~l = zR~x mod N if and only if z = «(REDCi»)-1 mod JV. The Jacobi symbol algorithm needs an extra negation if (R/N) = -1, since (xR~x/N) = (x/N)(R/N). Let M|N. A change of modulus from N (using R = R(N)) to M (using R = R(M)) proceeds normally if R(M) = R(N). If R(M) ¥= R(N), multiply each jV-residue by (R(N)/R(M))~x mod M during the conversion.

III. EXISTING AND PROPOSED SYSTEM

A.*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-to two (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) 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.

Multiplier	FC	Area Complexity	Area Ratio	Critical Path Delay	Delay Ratio
SCS-MM-1 [5]	Yes	$2bxd_{FA}+4bxd_{MUX}+bxd_{FF}+d_{CPA\_FC}$	$6.76bxd_{FA}$*	$\max(2xT_{AND}+2xT_{FA}, t(CPA\_FC))$	$2.68xT_{FA}$*
SCS-MM-2 [8]	Yes	$2bxd_{FA}+4bxd_{MUX}+bxd_{FF}+2bxd_{AND}+3bxd_{MUX}$	$8.24bxd_{FA}$	$2xT_{MUX}+T_{AND}+2xT_{FA}$	$3.24xT_{FA}$
FCS-MM-1 [9]	No	$3bxd_{FA}+5bxd_{MUX}+2bxd_{FF}+3bxd_{AND}$	$10.48bxd_{FA}$	$2xT_{MUX}+T_{AND}+3xT_{FA}$	$4.02xT_{FA}$
FCS-MM-2 [9]	No	$2bxd_{FA}+7bxd_{MUX}+2bxd_{FF}+2bxd_{AND}$	$12.56bxd_{FA}$	$2xT_{MUX}+T_{AND}+T_{MUX}+2xT_{FA}$	$3.73xT_{FA}$

Table I. Analysis of area & delay of different designs

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 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.

Cell	FA	REG	2-input NAND	2-input NOR	3-input NAND	3-input NOR	2-input AND	2-input XOR	3-input XOR	2-to-1* MUX	2-to-1 MUX	3-to-1 MUX	4-to-1 MUX
Area ratio	1.00	0.88	0.16	0.16	0.20	0.20	0.20	0.32	0.68	0.32	0.36	0.72	0.96
Delay ratio	1.00	-	0.12	0.16	0.20	0.32	0.34	0.34	0.95	0.23	0.45	0.63	0.71

Table. II Normalized area and delay of the standard cells 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 significantly 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.

Algorithm FCS-MM-1:
FCS-based Montgomery multiplication

Inputs : AS, AC, BS, BC, N (modulus)
Outputs : $SS[k+2], SC[k+2]$

1. $SS[0] = 0;\ SC[0] = 0;$
2. for $i = 0$ to $k + 1$ {
3. $q_i = (SS[i]_0 + SC[i]_0 + A_i \times (BS_0 + BC_0)) \bmod 2;$
4. $(SS[i+1], SC[i+1]) = (SS[i] + SC[i] + A_i \times (BS + BC) + q_i \times N) / 2;$
5. }
6. return $SS[k+2], SC[k+2]$;

Fig 5: *FCS-MM-1 Montgomery multiplication algorithm*

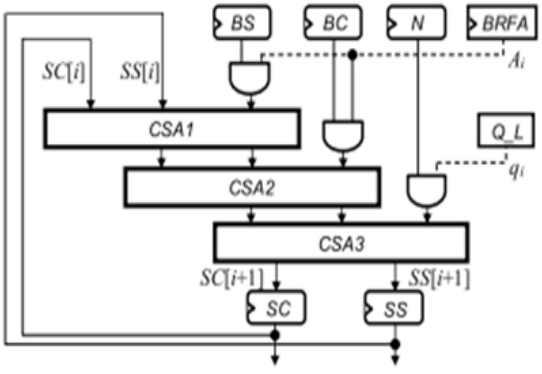


Fig 6: *FCS-MM-1 multiplier*

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.

B. *Critical Path Delay Reduction*

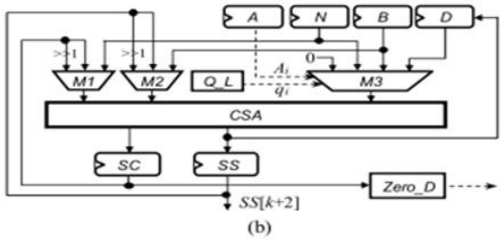
The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM-2. That is, we can pre compute D = B + N and reuse the one-level CSA architecture to perform B+N and the format conversion. Fig. 7(a) and (b) shows the modified SCS-based Montgomery multiplication (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 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. Therefore, the critical path delay of the MSCS-MM multiplier can be reduced into TMUX4 + TFA. However, in addition to performing the 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. 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.

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*

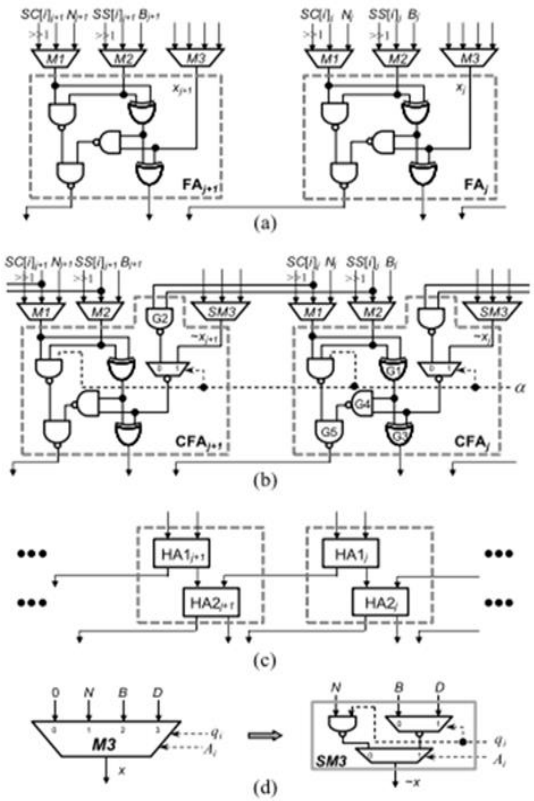


Fig 8*(a) Conventional FA circuit. (b) Proposed CFA Circuit. (c) Two serial HAs. (d) Simplified multiplexer SM3*

C. *Proposed Algorithm and Hardware Architecture*

On the bases of critical path delay reduction, clock cycle number reduction, and quotient pre computation mentioned above, a new SCS-based Montgomery MM algorithm (i.e., SCS-MM-New algorithm shown in Fig. 10) using one-level CCSA architecture is proposed to significantly reduce the required clock cycles for completing one MM. As shown in SCS-MM-New algorithm, steps 1–5 for producing Bˆ and Dˆ are first performed. Note that because qi+1 and qi+2 must be generated in the ith iteration, the iterative index i of Montgomery MM will start from −1 instead of 0 and the corresponding initial values of qˆ and Aˆ must be set to 0. Furthermore, the original for loop is replaced with the while loop in SCS-MM-New algorithm to skip some unnecessary iterations when skipi+1 = 1. In addition, the ending number of iterations in SCS-MM-New algorithm is changed to k + 4 instead of k + 1 in Fig. 7(a).

This is because B is replaced with Bˆ and thus three extra iterations for computing division by two are necessary to ensure the correctness of Montgomery MM. In the while loop, steps 8–12 will be performed in the proposed one-level CCSA architecture with one 4-to-1 multiplexer. The computations of qi+1, qi+2, and skipi+1 in step 13 and the selections of Aˆ, qˆ, and i in steps 14–20 can be carried out in parallel with steps 8–12. Note that the right-shift operations of steps 12 and 15 will be delayed to next clock cycle to reduce the critical path delay of corresponding hardware architecture. The hardware architecture of SCS-MM-New algorithm, denoted as SCS-MM-New multiplier, are shown in Fig. 11, which consists of one one-level CCSA architecture, two 4-to-1 multiplexers (i.e., M1 and M2), one simplified 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.

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 neither 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 SS1 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. Note that SC[i] 2:0 and SS[i] 2:0 can also be obtained from M1 and M2 but a longer delay is required because they are 4-to-1 multiplexers. After the while loop in steps 7–21 is completed, qˆ and Aˆ stored in FFs are reset to 0. Then, the format conversion in steps 23 and 24 can be performed by the SCS-MM-New multiplier similar to the computation of Dˆ = Bˆ + Nˆ in steps 3 and 4. Finally, SS [k + 5] in binary format is outputted when SC [k + 5] is equal to 0.

Algorithm SCS-MM-New:
Proposed SCS-based Montgomery multiplication

Inputs : A, B, $\hat{N}$ (new modulus)
Output : $SS[k+5]$

1. $\hat{B} = B << 3$; $\hat{q} = 0$; $\hat{A} = 0$; $skip_{i+1} = 0$;
2. $(SS, SC) = 1F\_CSA(\hat{B}, \hat{N}, 0)$;
3. while (SC != 0)
4. $(SS, SC) = 2H\_CSA(SS, SC)$;
5. $\hat{D} = SS$;
6. $i = -1$; $SS[-1] = 0$; $SC[-1] = 0$;
7. while ($i \le k + 4$) {
8. if ($\hat{A} = 0$ and $\hat{q} = 0$) $x = 0$;
9. if ($\hat{A} = 0$ and $\hat{q} = 1$) $x = \hat{N}$;
10. if ($\hat{A} = 1$ and $\hat{q} = 0$) $x = \hat{B}$;
11. if ($\hat{A} = 1$ and $\hat{q} = 1$) $x = \hat{D}$;
12. $(SS[i+1], SC[i+1]) = 1F\_CSA(SS[i], SC[i], x)>>1$;
13. compute q_{i+1}, q_{i+2}, and $skip_{i+1}$ by (5), (7) and (8);
14. if ($skip_{i+1} = 1$){
15. $SS[i+2] = SS[i+1]>>1$; $SC[i+2] = SC[i+1]>>1$;
16. $\hat{q} = q_{i+2}$; $\hat{A} = A_{i+2}$; $i = i + 2$;
17. }
18. else{
19. $\hat{q} = q_{i+1}$; $\hat{A} = A_{i+1}$; $i = i + 1$;
20. }
21. }
22. $\hat{q} = 0$; $\hat{A} = 0$;
23. while ($SC[k+5]$!= 0)
24. $(SS[k+5], SC[k+5]) = 2H\_CSA(SS[k+5], SC[k+5])$;
25. return $SS[k+5]$;

Fig 10.*SCS-MM New algorithm*

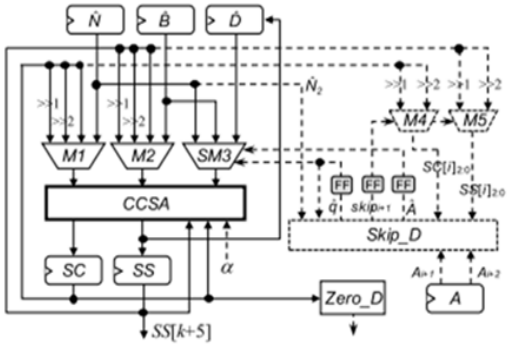


Fig 11.*SCS-MM New multiplier*

IV. SIMULATION TOOLS

Schematic diagrams of SCS based Montgomery modular multiplication:

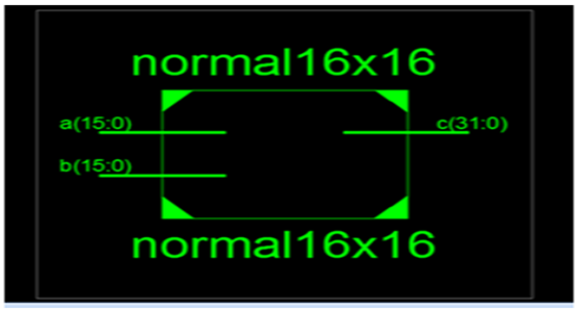


Fig 12 *Block diagram of SCS –MM*

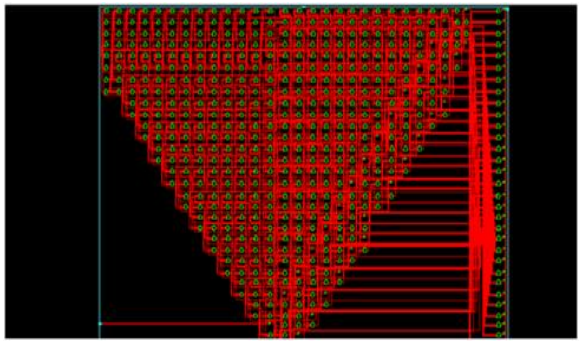


Fig 13 *Technology schematic diagram of SCS-MM*

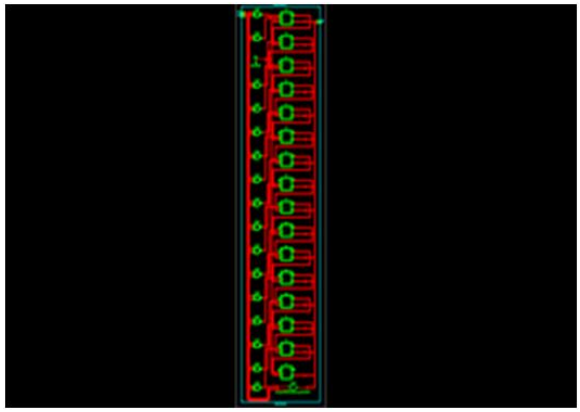


Fig 14 *schematic diagram of SCS-MM*

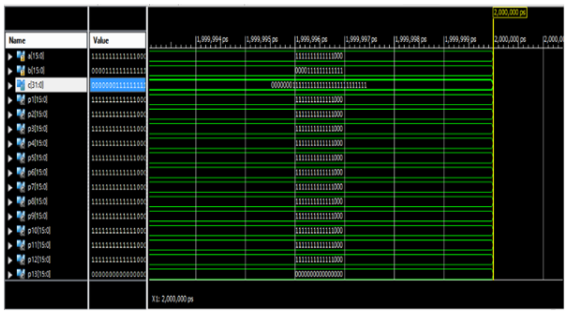


Fig 15 *Simulation result of SCS based Montgomery modular multiplication*

V. 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.

REFERENCE

[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.

[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, pp. 1136–1146, Jul. 2011.

[16] S.-H. Wang, W.-C. Lin, J.-H. Ye, and M.-D. Shieh, "Fast scalable radix-4 Montgomery modular multiplier," in Proc. IEEE Int. Symp. Circuits Syst., May 2012, pp. 3049–3052.

[17] J.-H. Hong and C.-W. Wu, "Cellular-array modular multiplier for fast RSA public-key cryptosystem based on modified Booth's algorithm," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 11, no. 3, pp. 474–484, Jun. 2003