Low-Cost High-Performance VLSI Architecture for Montgomery Modular Multiplication

Shiann-Rong Kuang, *Member, IEEE*, Kun-Yi Wu, and Ren-Yao Lu

***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 precomputation 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 precomputation 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 precomputation 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.**

I. 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 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.

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

Fig. 1. MM algorithm.

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

Manuscript received July 31, 2014; revised November 1, 2014 and January 10, 2015; accepted February 17, 2015. Date of publication March 13, 2015; date of current version January 19, 2016.

The authors are with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung 804, Taiwan (e-mail: srkuang@cse.nsysu.edu.tw; d963040012@student.nsysu.edu.tw; m013040038@gmail.com).

Digital Object Identifier 10.1109/TVLSI.2015.2409113

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.

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 precompute 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.

The remainder of this paper is organized as follows. Section II briefly reviews several radix-2 Montgomery MM algorithms. In Section III, we propose a simple and efficient SCS-based Montgomery MM algorithm and its hardware architecture. The comparisons of different Montgomery multipliers are made in Section IV. Finally, the conclusion is drawn in Section V.

II. Modular Multiplication Algorithms

A. Montgomery Multiplication

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 X_i in Fig. 1 shows the ith bit of X in binary representation. In addition, the notation $X_{i:j}$ 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 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 $2^{k+2} \bmod N$, 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 $S[k+2]$;

Fig. 2. SCS-based Montgomery multiplication algorithm.

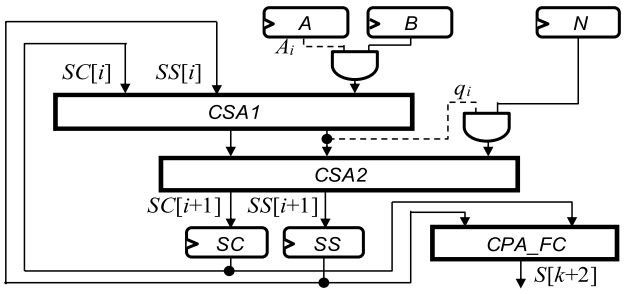


Fig. 3. SCS-MM-1 multiplier.

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] precomputed $D = B + N$ so that the computation of $A_i \times B + q_i \times N$ in step 4 of Fig. 2 can be simplified into one selection operation. One of the

TABLE I
ANALYSIS OF AREA AND DELAY OF DIFFERENT DESIGNS

Multiplier	FC	Area Complexity	Area Ratio	Critical Path Delay	Delay Ratio
SCS-MM-1 [5]	Yes	$2k\times A_{FA} + 4k\times A_{REG} + k\times A_{SR} + A_{CPA\_FC}$	$6.76k\times A_{FA}$*	$max(2\times T_{AND} + 2\times T_{FA}, \tau(CPA\_FC))$	$2.68\times T_{FA}$*
SCS-MM-2 [8]	Yes	$2k\times A_{FA} + 4k\times A_{REG} + k\times A_{SR} + 2k\times A_{AND} + 3k\times A_{MUX2}$	$8.24k\times A_{FA}$	$2\times T_{MUX2} + T_{AND} + 2\times T_{FA}$	$3.24\times T_{FA}$
FCS-MM-1 [9]	No	$3k\times A_{FA} + 5k\times A_{REG} + 2k\times A_{SR} + 3k\times A_{AND}$	$10.48k\times A_{FA}$	$2\times T_{XOR} + T_{AND} + 3\times T_{FA}$	$4.02\times T_{FA}$
FCS-MM-2 [9]	No	$2k\times A_{FA} + 7k\times A_{REG} + 2k\times A_{SR} + 2k\times A_{MUX4}$	$12.56k\times A_{FA}$	$2\times T_{XOR} + T_{AND} + T_{MUX4} + 2\times T_{FA}$	$3.73\times T_{FA}$

*: the ratio does not consider the extra CPA_FC

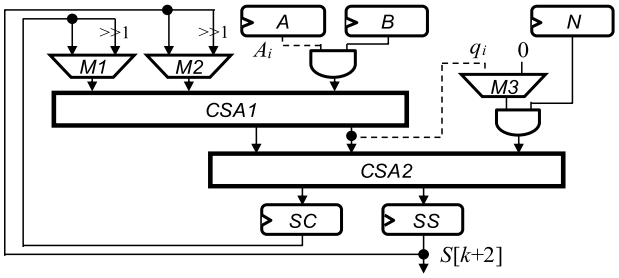


Fig. 4. SCS-MM-2 multiplier.

operands 0, N, B, and D will be chosen if $(A_i, q_i) = (0, 0)$, (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].

C. 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)

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

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

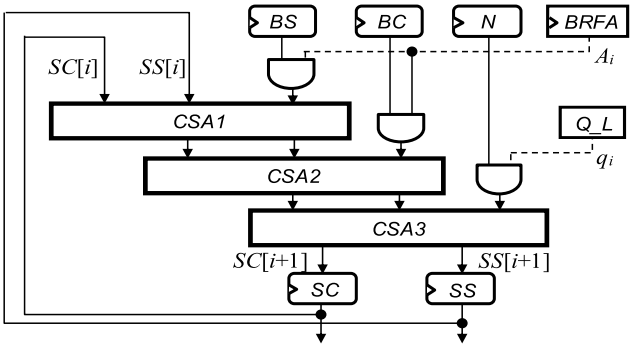


Fig. 6. FCS-MM-1 multiplier.

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 A_i and q_i 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

TABLE II
NORMALIZED AREA AND DELAY OF THE STANDARD CELLS

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* MUXI	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	0.93	0.23	0.45	0.63	0.71

*: the MUXI cell is a multiplexer with inverted output

notations A_G and T_G denote the area and delay of a cell G, respectively, and $\tau(\Omega)$ denotes the critical path delay of circuit Ω. Note that A_{SR} in Table I denotes the area of a shift register, and we assume that A_{SR} is approximate to the sum of A_{REG} and A_{MUX2}. 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.

III. 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.

A. Critical Path Delay Reduction

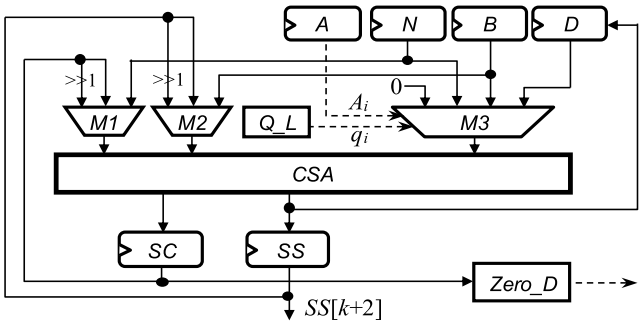
The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM-2. That is, we can precompute $D = B + N$ and reuse the one-level CSA architecture to perform $B+N$ and the format conversion. Fig. 7(a) and (b) shows the modified SCS-based Montgomery multiplication (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 q_i 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 A_i and q_i 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 $T_{MUX4} + T_{FA}$. 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 $SS = 111\ldots111_2$ and $SC = 000\ldots001_2$, the one-level CSA architecture needs k clock cycles to complete $SS + SC$.

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.

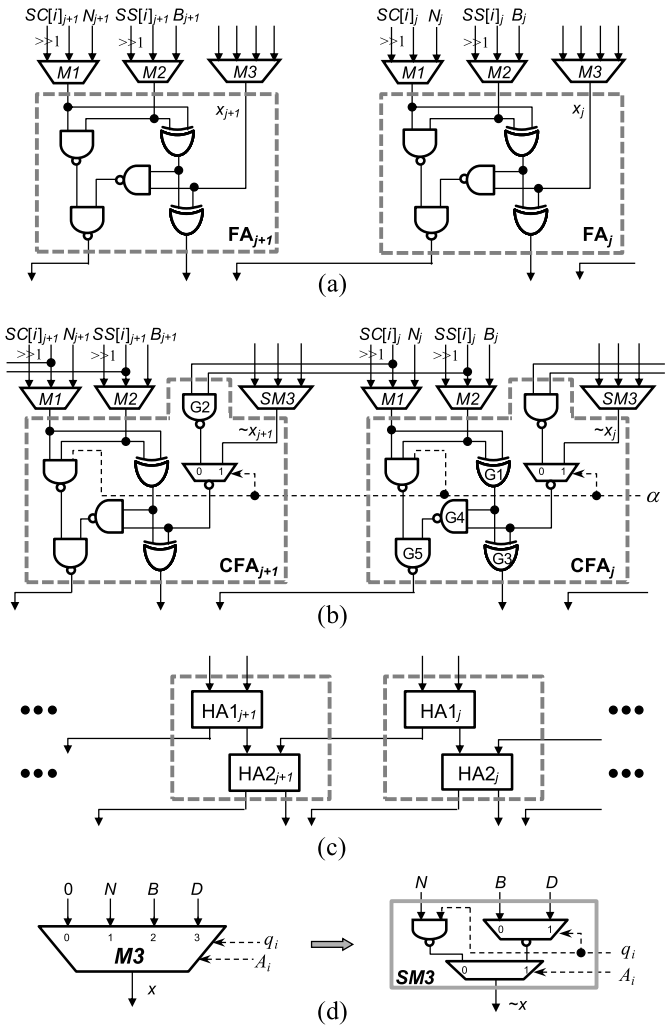


Fig. 8. (a) Conventional FA circuit. (b) Proposed CFA circuit. (c) Two serial HAs. (d) Simplified multiplexer *SM3*.

That is, $\sim 3k$ 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.

B. 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 CFA_j and G2 of CFA_{j+1} in Fig. 8(b) will act as HA1_j in Fig. 8(c), and G3, G4, and G5 of CFA_j 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 $\sim$ 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, T_{SM3} (i.e., $0.68 \times T_{\text{FA}}$) is approximate to T_{MUX3} (i.e., $0.63 \times T_{\text{FA}}$) and T_{MUXI2} (i.e., $0.23 \times T_{\text{FA}}$) is smaller than T_{XOR2} (i.e., $0.34 \times T_{\text{FA}}$). 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 $(SS, SC) = \text{2H\_CSA}(SS, SC)$ 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:

$$(SS[i+1], SC[i+1]) = (SS[i] + SC[i] + x)/2 \qquad (1)$$

where the variable x may be 0, N, B, or D depending on the values of A_i and q_i. The computation process of (1) is shown in Fig. 9. When $A_i = 0$ and $q_i = 0$, x is equal to 0 and $SS[i]_0$ must be equal to $SC[i]_0$ because the sum of $SS[i]_0 + SC[i]_0 + x_0$ is equal to 0. That is, if $A_i = 0$ and $q_i = 0$, 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 $A_i = q_i = 0$ 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 $A_i = q_i = 0$ and $SS[i]_0 = SC[i]_0 = 0$.

Accordingly, the signal skip_{i+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

$$\text{skip}_{i+1} = \sim (A_{i+1} \vee q_{i+1} \vee SS[i+1]_0) \qquad (2)$$

where $\vee$ represents the OR operation. If skip_{i+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, q_{i+1} and A_{i+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., $\text{skip}_{i+1} = 1$), $SS[i+1]$ 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 q_{i+1} and A_{i+1} but also q_{i+2} and A_{i+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 $\hat{q}$ and $\hat{A}$) for choosing the proper value of x in the next clock cycle must be picked from (q_{i+1}, A_{i+1}) or (q_{i+2}, A_{i+2}) according to the skip_{i+1} signal produced in the ith iteration. That is, $(\hat{q}, \hat{A}) = (q_{i+2}, A_{i+2})$ if $\text{skip}_{i+1} = 1$. Otherwise, $(\hat{q}, \hat{A}) = (q_{i+1}, A_{i+1})$.

C. Quotient Precomputation

As mentioned above, A_{i+1}, A_{i+2}, q_{i+1}, and q_{i+2} must be known in the ith iteration for skipping the unnecessary operation in the $(i+1)$th iteration. It is easy to obtain A_{i+1} and A_{i+2} in the ith iteration. The quotient q_{i+1} can be computed in the ith iteration similar to step 7 of Fig. 7(a) as follows:

$$q_{i+1} = (SS[i+1]_0 + SC[i+1]_0 + A_{i+1} \times B_0) \bmod 2. \quad (3)$$

However, $SS[i+1]_0$ and $SC[i+1]_0$ are unavailable until (1) is completed, as shown in Fig. 9. Therefore, the critical path of Montgomery multiplier in Fig. 7(b) will be largely lengthened if (3) is directly used to produce q_{i+1} in the ith iteration. To avoid this situation, N, B, and D are modified as follows so that $SS[i+1]_0$, $SC[i+1]_0$, q_{i+1}, and q_{i+2} can be quickly generated in the ith iteration.

Since modulus N is an odd number and is added in the ith iteration only when q_i is equal to one, it is found that at least a propagated carry 1 is generated since N_0 is equal to one. Therefore, we can directly employ the value $\hat{N}$ as shown in (4) instead of N to accomplish the process of Montgomery MM. Afterward, $\hat{N}_{1:0}$ must be equal to zero

$$\hat{N} = \begin{cases} N+1, & \text{if } N_{1:0} = 11 \\ 3N+1, & \text{if } N_{1:0} = 01. \end{cases} \quad (4)$$

Moreover, we employ $\hat{B} = 8B$ instead of B to ensure that $\hat{B}_{2:0}$ is equal to zero so that $A_{i+1} \times B_0$ in (3) can be eliminated and the computation of q_{i+2} can also be simplified. Note that three extra clock cycles at the end of MM for computing division by two are necessary to maintain the correctness of Montgomery MM because B is replaced with $8B$. If N and B are replaced by $\hat{N}$ and $\hat{B}$, the produced $\hat{D}_{1:0}$ ($\hat{D} = \hat{N} + \hat{B}$) must be equal to zero.

After N, B, and D are replaced by $\hat{N}$, $\hat{B}$, and $\hat{D}$, we can ensure that the two LSBs of variable x (i.e., $x_{1:0}$) in (1) must be equal to zero. As a result, the carry value $SC[i+1]_0$ in Fig. 9 is equal to $SS[i]_0 \wedge SC[i]_0$ since $x_0 = 0$, where $\wedge$ denotes the AND operation. Moreover, the sum value $SS[i+1]_0$ in Fig. 9 is equal to $SS[i]_1 \oplus SC[i]_1$ because $x_1 = 0$, where $\oplus$ is the XOR operation. According to the above results, the logic expression in (3) for generating q_{i+1} in the ith iteration can be rewritten as

$$q_{i+1} = (SS[i]_1 \oplus SC[i]_1) \oplus (SS[i]_0 \wedge SC[i]_0). \quad (5)$$

Similar to (3), the quotient q_{i+2} can be generated in the ith iteration by the following equation:

$$q_{i+2} = (SS[i+2]_0 + SC[i+2]_0) \bmod 2. \quad (6)$$

The q_{i+2} will be selected in the ith iteration only when $\text{skip}_{i+1} = 1$. In this case, $A_{i+1} = q_{i+1} = 0$ and $SS[i+1]_0 = SC[i+1]_0 = 0$ so that $SS[i+2]_0 + SC[i+2]_0$ in Fig. 9 is equal to $SS[i+1]_1 + SC[i+1]_1$. Because $x_1 = 0$, $SC[i+1]_1 = SS[i]_1 \wedge SC[i]_1$. Moreover, $SS[i+1]_1$ is equal to $SS[i]_2 \oplus SC[i]_2 \oplus x_2$ and x_2 is equal to 0, $\hat{N}_2$, 0, or $\hat{N}_2$ when $(A_i, q_i) = (0, 0)$, $(0, 1)$, $(1, 0)$, or $(1, 1)$. Therefore, we can obtain that $x_2 = q_i \wedge \hat{N}_2$. As a result, (6) can be simplified and expressed as

$$q_{i+2} = (SS[i]_2 \oplus SC[i]_2) \oplus (q_i \wedge \hat{N}_2) \oplus (SS[i]_1 \wedge SC[i]_1). \quad (7)$$

In addition to q_{i+1} and q_{i+2} can be simplified into (5) and (7), we can also derive a simpler expression for skip_{i+1} in (2). Let $\delta_1 = SS[i]_1 \oplus SC[i]_1$ and $\delta_0 = SS[i]_0 \wedge SC[i]_0$, then

$$\begin{aligned} \text{skip}_{i+1} &= \sim (A_{i+1} \vee q_{i+1} \vee SS[i+1]_0) \\ &= \sim (A_{i+1} \vee (\delta_1 \oplus \delta_0) \vee \delta_1) \\ &= \sim (A_{i+1} \vee \delta_1 \vee \delta_0) \\ &= \sim (A_{i+1} \vee (SS[i]_1 \oplus SC[i]_1) \vee (SS[i]_0 \wedge SC[i]_0)). \end{aligned} \quad (8)$$

According to (8), we can quickly obtain skip_{i+1} in the ith iteration by $SS[i]_1$, $SC[i]_1$, $SS[i]_0$, and $SC[i]_0$.

D. Proposed Algorithm and Hardware Architecture

On the bases of critical path delay reduction, clock cycle number reduction, and quotient precomputation 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 $\hat{B}$ and $\hat{D}$ are first performed. Note that because q_{i+1} and q_{i+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 $\hat{q}$ and $\hat{A}$ 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 $\text{skip}_{i+1} = 1$. 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 $\hat{B}$ 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 q_{i+1}, q_{i+2}, and skip_{i+1} in step 13 and the selections of $\hat{A}$, $\hat{q}$, 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

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.

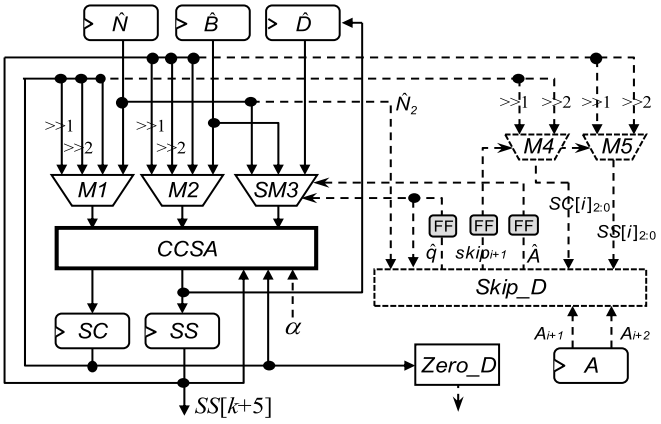


Fig. 11. SCS-MM-New multiplier.

multiplier *SM3*, one skip detector *Skip_D*, one zero detector *Zero_D*, and six registers. *Skip_D* is developed to generate skip_{i+1}, $\hat{q}$, and $\hat{A}$ 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.

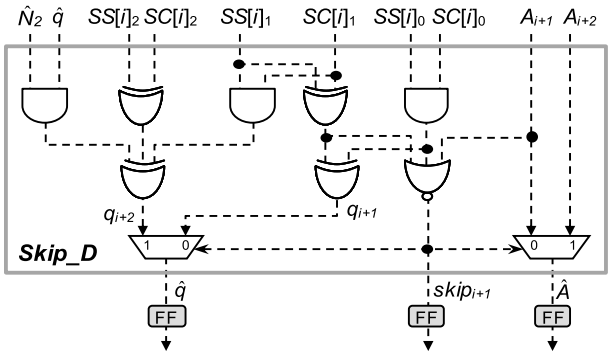


Fig. 12. Skip detector *Skip_D*.

At the beginning of Montgomery multiplication, the FFs stored skip_{i+1}, $\hat{q}$, $\hat{A}$ are first reset to 0 as shown in step 1 of SCS-MM-New algorithm so that $\hat{D} = \hat{B} + \hat{N}$ 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 skip_{i+1}, $\hat{q}$, and $\hat{A}$. 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 q_{i+1}, q_{i+2}, and skip_{i+1} signal in the ith iteration according to (5), (7), and (8), respectively, and then selects the correct $\hat{q}$ and $\hat{A}$ according to skip_{i+1}. At the end of the ith iteration, $\hat{q}$, $\hat{A}$, and skip_{i+1} must be stored to FFs. In the next clock cycle of the ith iteration, *SM3* outputs a proper x according to $\hat{q}$ and $\hat{A}$ generated in the ith iteration as shown in steps 8–11, and *M1* and *M2* output the correct *SC* and *SS* according to skip_{i+1} generated in the ith iteration. If $\text{skip}_{i+1} = 0$, $SC \gg 1$ and $SS \gg 1$ are selected. Otherwise, $SC \gg 2$ and $SS \gg 2$ 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 skip_{i+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, $\hat{q}$ and $\hat{A}$ 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 $\hat{D} = \hat{B} + \hat{N}$ in steps 3 and 4. Finally, $SS[k+5]$ in binary format is outputted when $SC[k+5]$ is equal to 0.

IV. Experimental Results

In this section, we first analyze the critical path delay and area of the proposed SCS-MM-New multiplier according to the information listed in Table II. Then, the delay and area are compared with that of previous designs. In addition, the average clock cycles of different Montgomery multipliers to complete one MM operation are also measured. Finally, several Montgomery multipliers are implemented and synthesized to demonstrate the efficiency of the proposed approach.

A. Analysis of Delay, Area, and Clock Cycle Number

As shown in Figs. 11 and 12, the maximum delay for generating $\hat{q}$ through *M4*, *M5*, and *Skip_D* is

TABLE III
COMPARISONS OF DIFFERENT MONTGOMERY MULTIPLIERS WITH 1024- AND 2048-BIT KEY SIZES

Key Size	Multiplier	#Cycle	ΔC (%)	Delay (*ns*)	Area (μm^2)	Time (μs)	Throughput Rate (*Mbps*)	ATP ($10^3\ \mu m^2 \times \mu s$)
1024	SCS-MM-1(32) [5]	1072	+4.2	4.93	487171	5.2850	193.8	2574.68
	SCS-MM-2 [8]	1049	+1.9	5.60	406127	5.8744	174.3	2385.75
	FCS-MM-1 [9]	1029	-	5.80	518214	5.9682	171.6	3092.80
	FCS-MM-2 [9]	1029	-	6.01	677121	6.1843	165.6	4187.51
	FCS-MMM42 [10]	822	−20.1	5.56	749076	4.5703	224.1	3423.52
	SCS-MM-New	880	−14.5	4.00	498379	3.5200	290.9	1754.29
2048	SCS-MM-1(32) [5]	2128	+3.7	5.28	956484	11.2358	182.3	10746.90
	SCS-MM-1(64) [5]	2096	+2.1	6.53	660156	13.6869	149.6	9035.48
	SCS-MM-2 [8]	2071	+0.9	5.92	732502	12.2603	167.0	8980.71
	FCS-MM-1 [9]	2053	-	6.00	1007807	12.3180	166.3	12414.17
	FCS-MM-2 [9]	2053	-	5.96	1347114	12.2359	167.4	16483.13
	FCS-MMM42 [10]	1636	−20.3	5.68	1448068	9.2925	220.4	13456.14
	SCS-MM-New	1734	−15.5	4.41	950184	7.6469	267.8	7266.00

$T_{\mathrm{MUX2}} + T_{\mathrm{XOR2}} + T_{\mathrm{XOR3}} + T_{\mathrm{MUX2}}$. According to Table II, the maximum delay for generating $\hat{q}$ will be $2.17 \times T_{\mathrm{FA}}$. Moreover, T_{SM3} and T_{MUXI2} are less than T_{MUX4} and T_{XOR2}, respectively, as mentioned in Section III-B. Therefore, the maximum delay of the one-level CCSA architecture in Fig. 11 for generating *SS* and *SC* is approximate to $T_{\mathrm{MUX4}} + T_{\mathrm{FA}}$ (i.e., $1.71 \times T_{\mathrm{FA}}$). As a result, the critical path delay of the SCS-MM-New multiplier is $2.17 \times T_{\mathrm{FA}}$, which is less than that of all Montgomery multipliers listed in Table I. Furthermore, the critical path delay of the FCS-MMM42 multiplier in [10] is $T_{\mathrm{MUX4}} + 2 \times T_{\mathrm{FA}}$ (i.e., $2.71 \times T_{\mathrm{FA}}$), which is much longer than that of the proposed SCS-MM-New multiplier.

On the other hand, the area complexity of SCS-MM-New multiplier is $k \times A_{\mathrm{CFA}} + 5k \times A_{\mathrm{REG}} + k \times A_{\mathrm{SR}} + 2k \times A_{\mathrm{MUX4}} + k \times A_{SM3}$ according to Fig. 11. As shown in Fig. 8, FA is modified into CFA at the expense of one 2-input NAND gate and one 2-to-1 MUXI cell. Therefore, A_{CFA} is approximate to $A_{\mathrm{FA}} + A_{\mathrm{NAND2}} + A_{\mathrm{MUXI2}}$. In addition, A_{SM3} is approximate to $A_{\mathrm{NAND2}} + A_{\mathrm{MUXI2}} + A_{\mathrm{MUX2}}$, as shown in Fig. 8(d). As a result, the area complexity of the proposed SCS-MM-New multiplier will be approximate to $9.88k \times A_{\mathrm{FA}}$. When compared with the previous multipliers listed in Table I, the area of SCS-MM-New multiplier is larger than that of SCS-MM-2 multiplier, but smaller than that of FCS-MM-1 and FCS-MM-2 multipliers. In addition, the area complexity of the FCS-MMM42 multiplier in [10] is $2k \times A_{\mathrm{FA}} + 7k \times A_{\mathrm{REG}} + 2k \times A_{\mathrm{SR}} + 2k \times A_{\mathrm{MUX2}} + 2k \times A_{\mathrm{MUX4}}$ (i.e., $13.28k \times A_{\mathrm{FA}}$), which is much larger than that of the proposed SCS-MM-New multiplier. Finally, if the nonconfigurable one-level CSA architecture is used in the SCS-MM-New multiplier, its area complexity will be reduced to $9.4k \times A_{\mathrm{FA}}$. That is, ∼5% additional area is required to form the CCSA architecture.

To evaluate the average clock cycles for completing one Montgomery MM, the above-mentioned multipliers, including SCS-MM-1 [5], SCS-MM-2 [8], FCS-MM-1 [9], FCS-MM-2 [9], FCS-MMM42 [10], and the proposed SCS-MM-New with 1024- and 2048-bit key sizes were designed and specified in Verilog hardware description language. Note that, we apply a carry-lookahead adder (CLA) to carry out the format conversion in SCS-MM-1 because of its reduced critical path delay. Moreover, SCS-MM-1(32) and SCS-MM-1(64) denote that 32- and 64-bit CLA are adopted for format conversion, respectively. The average clock cycles of different multipliers for completing one Montgomery MM (denoted as #Cycle) are measured and shown in Table III through the simulation with 10 000 random input patterns. Note that ΔC in Table III denotes the cycle number increment when compared with FCS-MM-1 multiplier. As can be seen from Table III, SCS-MM-1 and SCS-MM-2 multipliers need more clock cycles than FCS-MM-1 multiplier. On the other hand, the proposed 1024- and 2048-bit SCS-MM-New multipliers offer cycle number reductions of 14.5% and 15.5% over the FCS-MM-1 multiplier, respectively. The FCS-MMM42 multiplier in [10] spends the least clock cycles because no extra clock cycles for format conversion are required.

B. Implementation Results

To further verify the efficiency of the proposed design, we synthesized those Montgomery modular multipliers listed in Table III by Synopsys Design Compiler with TSMC 90-nm CMOS cell library. Subsequently, the Cadence SoC Encounter was employed to perform the placement and routing. Delay estimations were obtained behind *RC* extraction from the placed and routed netlists. The implementation results, including the critical path delay (Delay), the hardware area (Area), the execution time (Time), and the throughput rate of these modular multipliers are given in Table III. The execution time is the required time to accomplish one Montgomery MM, i.e., #Cycle × Delay. The throughput rate is formulated as the key size multiplied by the frequency (the reciprocal of Delay) and then divided by #Cycle.

As the results shown in Table III, the proposed SCS-MM-New multiplier has the shortest critical path delay and needs fewer clock cycles to complete one Montgomery MM, and thus spends the least execution time and achieves the highest throughput rate. Note that the critical path delay of SCS-MM-1(64) is significantly lengthened by the

64-bit CPA_FC. On the other hand, the SCS-MM-2 multiplier generally has smaller area than other designs. The proposed SCS-MM-New multiplier also needs more area than the SCS-MM-2 multiplier due to extra multiplexers introduced to shorten the critical path delay and reduce the required clock cycles. Nevertheless, the area of the proposed SCS-MM-New multiplier is still less than that of FCS-based multipliers. As a consequence, SCS-MM-New can obtain the smallest ATP than previous radix-2 Montgomery multipliers. When compared with the FCS-MMM42 multiplier, the proposed 1024-bit (2048-bit) SCS-MM-New multiplier achieves 28.1% (22.4%) shorter critical path and 33.5% (34.4%) smaller hardware area, leading to 29.8% (21.5%) throughput enhancement and 48.8% (46.0%) ATP improvement. The results in Table III are consistent with the analyses in Section IV-A and show that the proposed approach is indeed capable of significantly enhancing the performance of radix-2 CSA-based Montgomery multiplier while maintaining low hardware complexity.

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.

Acknowledgment

The authors would like to thank the reviewers for their many constructive comments and suggestions in improving this paper. They would also like to thank the contributions of Taiwan Semiconductor Manufacturing Company Limited and National Chip Implementation Center, Taiwan, for their support in technology data.

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.

[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.

[18] F. Gang, "Design of modular multiplier based on improved Montgomery algorithm and systolic array," in *Proc. 1st Int. Multi-Symp. Comput. Comput. Sci.*, vol. 2. Jun. 2006, pp. 356–359.

[19] G. Perin, D. G. Mesquita, F. L. Herrmann, and J. B. Martins, "Montgomery modular multiplication on reconfigurable hardware: Fully systolic array vs parallel implementation," in *Proc. 6th Southern Program. Logic Conf.*, Mar. 2010, pp. 61–66.

[20] A. Cilardo, A. Mazzeo, L. Romano, and G. P. Saggese, "Exploring the design-space for FPGA-based implementation of RSA," *Microprocessors Microsyst.*, vol. 28, no. 4, pp. 183–191, May 2004.

[21] D. Bayhan, S. B. Ors, and G. Saldamli, "Analyzing and comparing the Montgomery multiplication algorithms for their power consumption," in *Proc. Int. Conf. Comput. Eng. Syst.*, Nov. 2010, pp. 257–261.

[22] C. D. Walter, "Montgomery exponentiation needs no final subtractions," *Electron. Lett.*, vol. 35, no. 21, pp. 1831–1832, Oct. 1999.

Shiann-Rong Kuang (M'09) received the B.S. degree from National Central University, Zhongli, Taiwan, in 1990, and the M.S. and Ph.D. degrees from National Cheng Kung University, Tainan, Taiwan, in 1992 and 1998, respectively, all in electrical engineering.

He is currently an Associate Professor with the Department of Computer Science and Engineering, National Sun Yat-sen University, Kaohsiung, Taiwan. His current research interests include low-power/high-performance VLSI architectures for public-key cryptosystems, and digital signal processing systems.

Kun-Yi Wu received the B.S. degree from Tunghai University, Taichung, Taiwan, in 2005, and the M.S. and Ph.D. degrees in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2007 and 2014, respectively.

He is currently a Senior Engineer with Nuvoton Technology Corporation, Kaohsiung. His current research interests include VLSI design, low-power design, heuristics, and optimization algorithms.

Ren-Yao Lu received the B.S. and M.S. degrees in computer science and engineering from National Sun Yat-sen University, Kaohsiung, Taiwan, in 2012 and 2014, respectively.

His current research interests include VLSI architectures for public-key cryptosystems and low-power design.