HIGH THROUGHPUT VLSI ARCHITECTURE FOR MONTGOMERY MODULAR MULTIPLICATION USING CARRY SAVE ADDER

R. Madhavi[1] madhavi.madhu03@gmail.com[1]

K. Bindhu Madhavi[2] bindum.ece@hitam.org[2]

K. Anil Kumar[3] anilkumar10436@gmail.com[3]

[1]PG. Scholar, Dept of ECE, Hyderabad Institute of Technology and Management, Gowdavelly, Medchal, Ranga Reddy, Hyderabad.

[2]Associate Professor, Dept of ECE, Hyderabad Institute of Technology and Management, Gowdavelly, Medchal, Ranga Reddy, Hyderabad.

[3]Assistant Professor, Dept of ECE, Hyderabad Institute of Technology and Management, Gowdavelly, Medchal, Ranga Reddy, Hyderabad.

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 operation 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.

I.INTRODUCTION

In Many public-key cryptosystems, 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 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×B×R−1 (mod N), where Nis the k-bit modulus, R −1 is the inverse of RmoduloN,and R=2k mod 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 approachesbased on carry-save addition were proposed to achieve a significant speedup of Montgomery MM.

```
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

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, 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. Finally, the condition and detection circuit, which are different with that of FCS-MMM42 multiplier, 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.

II. EXISTING SYSTEM

MODULARMULTIPLICATIONALGORITHMS

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$S=A\times B\times R^{-1}(\text{mod } N)$,where$R^{-1}$isthe inverse of RmoduloN.Thatis,$R\times R^{-1}=1(\text{mod}N)$. Note that, the notation Xi in Fig. 1 shows the ith bit of Xin 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 changed the number of iterations and the value of R to k+2 and 2k+2modN, 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 multiplicationalgorithm.

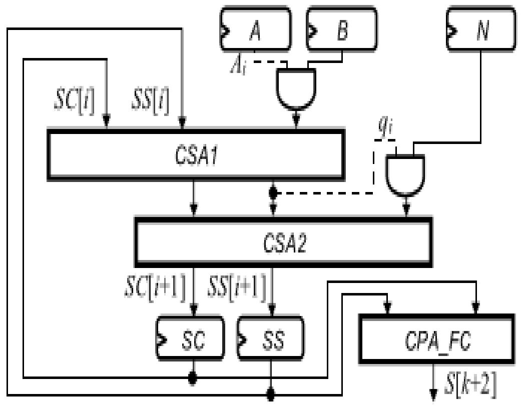


Fig.3. SCS-MM-1 multiplier

B. SCS-Based Montgomery Multiplication

`To avoid the long carry propagation, the intermediate results 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. 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 (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.

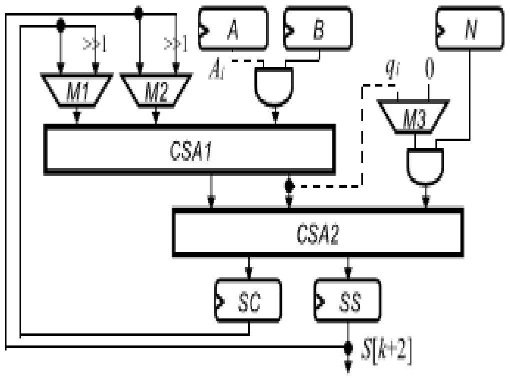


Fig.4. SCS-MM-2 multiplier

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.     $q_i = (SS[i]_0 + SC[i]_0 + A_i \times (BS_0 + BC_0)) \bmod 2$;
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

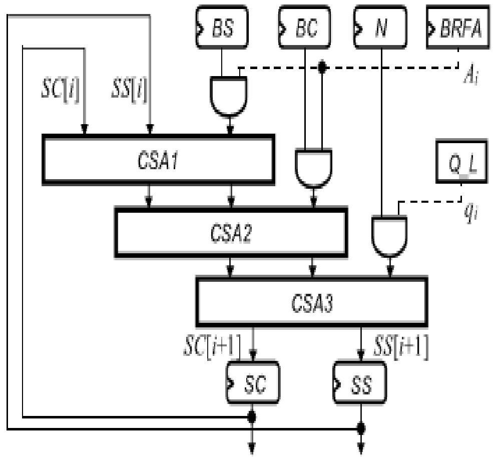


Fig.6.FCS-MM-1 multiplier.

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 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.

III. LITERATURE SURVEY:

Low power hardware design for montgomery Modular multiplication

This paper describes the design and implementation of low power modular multiplier of

RSA and balances its area and speed. By improving Montgomery modular multiplication algorithm, optimizing critical path and using several low power methods, this paper achieves low power as well as high speed performance. The design is implemented using SMIC 0.13um CMOS process, the average power consumption is 106uW at 13.56MHZ when executing 1024-bit operations, the area is about 0.17mm2and the time to finish modular multiplication are 1412 clock cycles, such excellent property make it suitable for RSA operation.

Novel Techniques for Montgomery Modular Multiplication Algorithms for Public Key Cryptosystems.

Extension of Montgomery multiplication algorithms in GF(p) are studied and analyzed. The time and space requirements of various state-of-the-art algorithms are presented. We propose Modified Montgomery Modular Multiplication Algorithms that reduces the number of computational operations such as number of additions, memory reads and writes involved in the existing algorithms, thereby, saving considerable time and area for execution. Many design examples has been solved to prove the theoretical correctness of the proposed algorithms. Complexity analysis shows that Modified Coarsely Integrated Scanning (MCIOS) consume less space and time compared to other modified Montgomery Algorithms. To verify the logical correctness, the proposed MCIOS algorithm was implemented in Xilinx Spartan3E FPGA. The total memory for execution of 64 –bit operand is 135484 KB for MCIOS and 140496 KB for existing Coarsely Integrated Scanning (CIOS) method. The proposed algorithm can be changed to be suitable for any arbitrary Galois field size with little modifications. Also the proposed algorithm can be developed as architecture suitable for System on Chip (SoC) implementations of Elliptic curve cryptosystem. Subsequently, the system can be developed as a 3D chip.

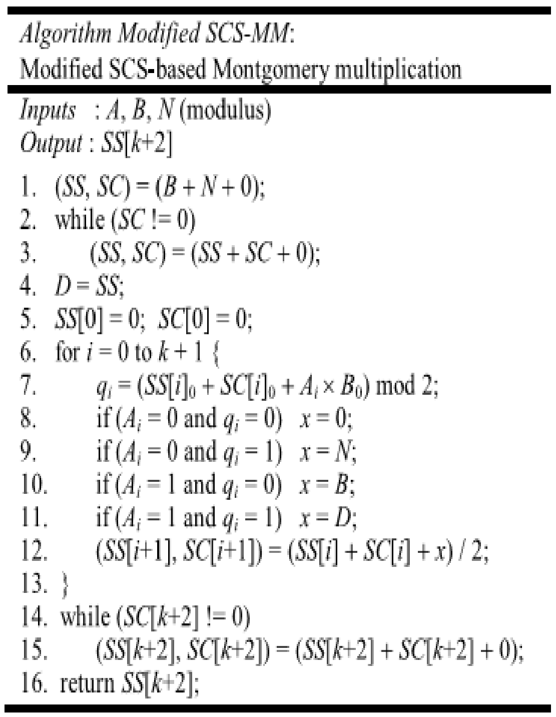
IV. PROPOSED SYSTEM

ProposedMontgomeryMultiplication

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 precomputeD=B+Nand 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). 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+Nand the format conversion via the one-level CSA architecture because they must be performed once in every MM.



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

(a)

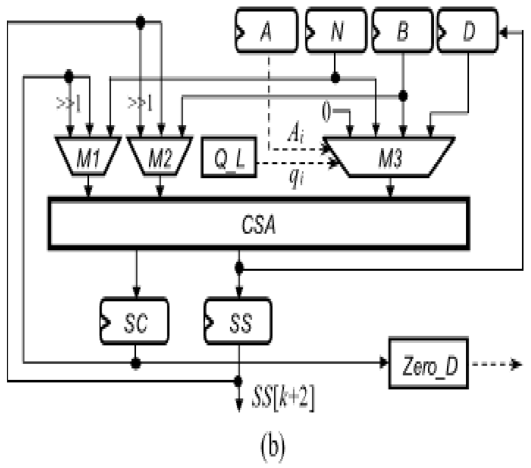


Fig.7.(a) Modified SCS-based Montgomery multiplication algorithm. (b) MSCS-MM multiplier

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 α =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 andG2ofCFAj+1in 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 HA2j in Fig. 8(c). Moreover, we modify the 4-to-1 multiplexer M3in Fig. 7(b) into a simplified multiplier SM3as shown in Fig. 8(d) because one of its inputs is zero, where~denotes the INVERT operation. Note that M3has been replaced bySM3in the proposed one-level CCSA architecture shown inFig. 8(b) According to the delay ratio shown in Table II,TSM3 (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, C)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 ashort 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.

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

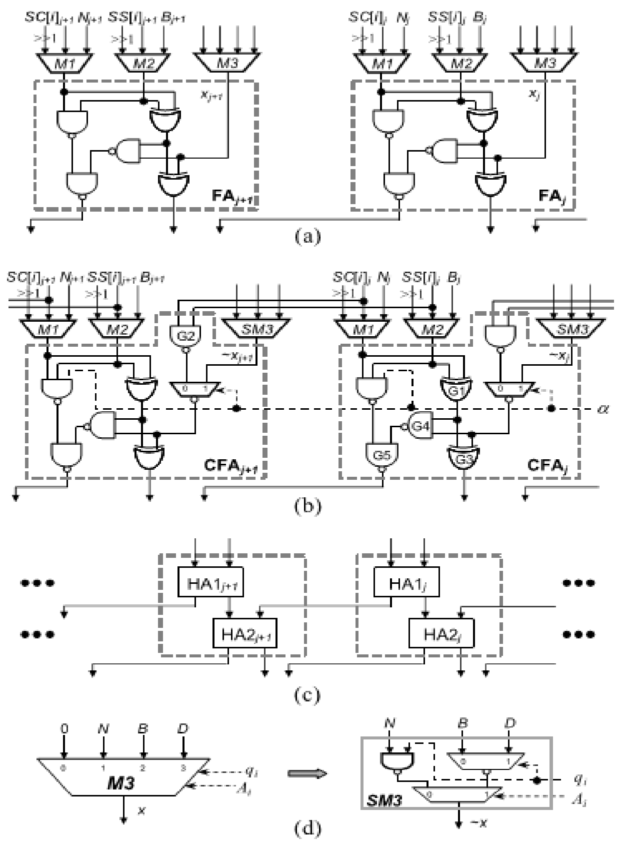


Fig.8. (a)Conventional FA circuit. (b) Proposed CFA circuit. (c) Two serial HAs. (d) Simplified multiplexerSM3

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 bitposition (i.e., divided by 2) when Ai = qi = 0and SS[i]0=SC[i]0=0.Accordingly, the signal skipi+1 used in theith iteration to indicate whether the carry-save addition in the (i +1)th iteration will be skipped can be expressed as

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

where∨represents the ORoperation. 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 theith iteration can be stored in FFs and then used to fast select the value of xin the (i+1)th iteration. Otherwise (i.e., skipi+1 =1),SS[i +1] andSC[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.

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

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

C. Quotient Precomputation:

As mentioned above,Ai+1, Ai+2, qi+1,andqi+2 must be known in theith iteration for skipping the unnecessary operation in the (i+1)th iteration. It is easy to obtainAi+1and Ai+2 in the ith iteration. The quotient qi+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]0andSC[i+1]0are 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 qi+1 in the ith iteration. To avoid this situation,N, B,andDare modified as follows so that SS[i +1]0, SC[i +1]0, qi+1,andqi+2 can be quickly generated in theith iteration. Since modulusNis an odd number and is added in the ith iteration only when qi is equal to one, it is found that at least a propagated carry 1 is generated since N0 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 $\hat{N}$:0 must be equal to zero.

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

According to the above results, the logic expression in (3) for generatingqi+1 in the ith iteration can be rewritten as

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

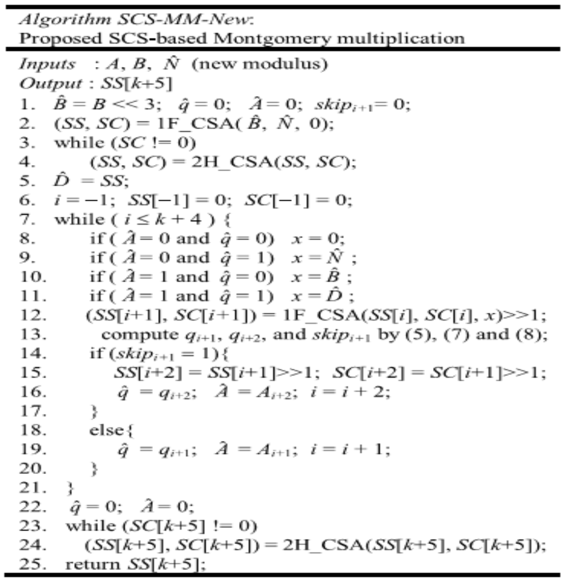
Similar to (3), the quotientqi+2 can be generated in the ith iteration by the following equation

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

Theqi+2 will be selected in the ith iteration only when skipi+1=1. In this case,Ai+1=qi+1=0andSS[i +1]0= SC[i +1]0 =0sothatSS[i +2]0+SC[i +2]0 in Fig. 9 is equal to SS[i +1]1 +SC[i +1]1. Because x1 =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 qi+1 and qi+2 must be generated in theith 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.



```
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

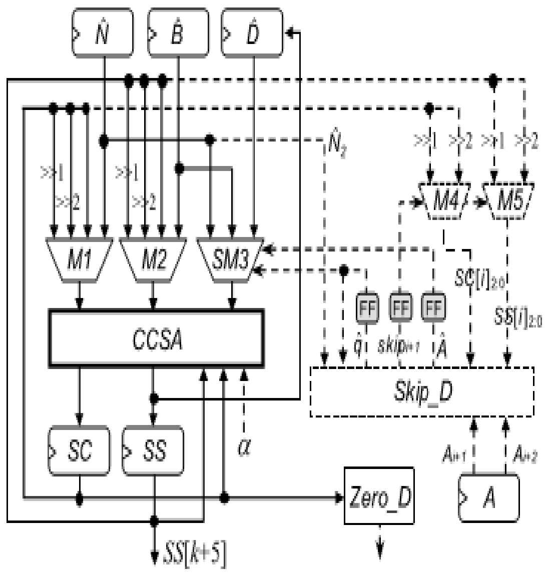


Fig.11.SCS-MM-New multiplier.

Multiplier SM3, one skip detector Skip_D, one zero detector Zero_D, and six registers. Skip_D is developed to generate skipi+1, $\hat{q}$, and $\hat{A}$ in the ith iteration. Both M4and M5 in Fig. 11 are 3-bit 2-to-1 multiplexers and they are much smaller thank-bit multiplexersM1, M2,andSM3. 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 M1andM2in Fig. 11 are generated by the control part, which are not depicted for the sake of simplicity.

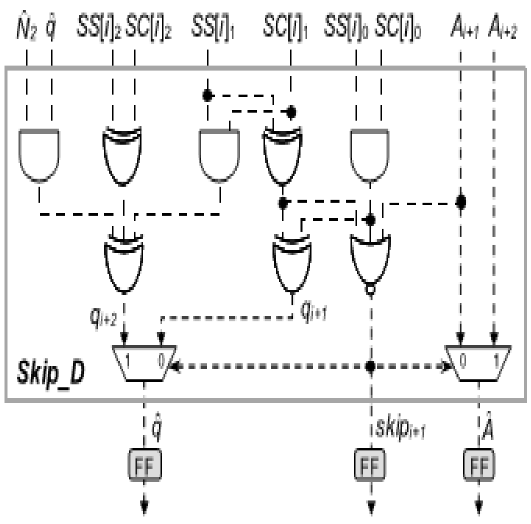
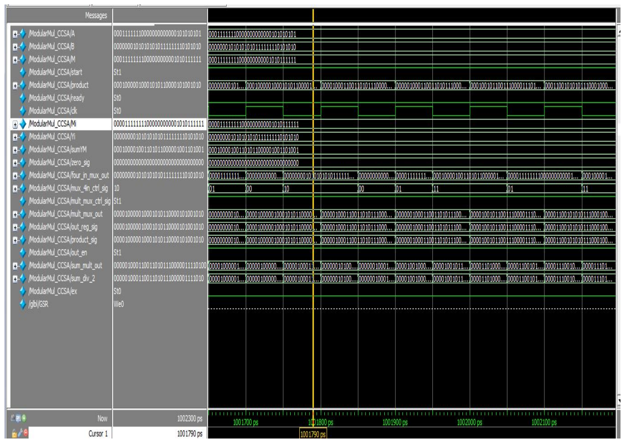


Fig.12. Skip detector Skip_D.

V. RESULTS

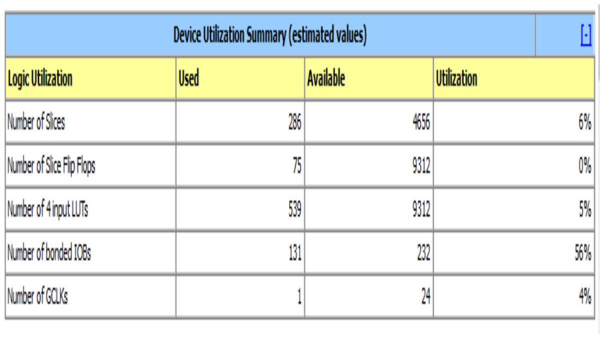
SIMULATION:



Synthesis Results:

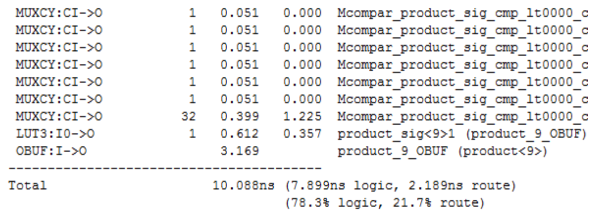
RTL Schematic:

DESIGN SUMMARY:



Device Utilization Summary (estimated values)			[-]
Logic Utilization	Used	Available	Utilization
Number of Slices	286	4656	6%
Number of Slice Flip Flops	75	9312	0%
Number of 4 input LUTs	539	9312	5%
Number of bonded IOBs	131	232	56%
Number of GCLKs	1	24	4%

SYNTHESIS REPORT:



```
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O           1   0.051   0.000  Mcompar_product_sig_cmp_lt0000_c
 MUXCY:CI->O          32   0.399   1.225  Mcompar_product_sig_cmp_lt0000_c
 LUT3:I0->O            1   0.612   0.357  product_sig<9>1 (product_9_OBUF)
 OBUF:I->O                 3.169          product_9_OBUF (product<9>)
----------------------------------------
Total                     10.088ns (7.899ns logic, 2.189ns route)
                                   (78.3% logic, 21.7% route)
```

VI. 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 inCryptology. 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 implementationof 1024-bit modular processor for RSA cryptosystem,” inProc. 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 crypto processor for RSA cryptosystem basedon modified Montgomery’s algorithm,” inProc. 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 CSAarchitecture for Montgomery modular multiplication,” Microprocessors Microsyst., vol. 31, no. 7, pp. 456–459, Nov. 2007.
[9] C. McIvor, M. McLoone, and J. V. Mc Canny, “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-efficienthigh-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-partitionmethod to perform Montgomery multiplication,” inProc. 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 ofradix-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-radixMontgomery multipliers,” inProc. 42nd Asilomar Conf. Signals, Syst.,Comput., Oct. 2008, pp. 772–776.