© 2017 IJSRSET | Volume 3 | Issue 3 | Print ISSN: 2395-1990 | Online ISSN : 2394-4099
Themed Section: Engineering and Technology

Modified VLSI Architecture for Montgomery Modular Multiplication

Bhagyamma S, A L Choodarathnakara, Vindya N D, Swamy Y T, Meghana H
Department of Electronics & Communication Engineering, GEC, Kushalnagar, Kodagu, Karnataka, India

ABSTRACT

This paper proposes a simple and efficient Montgomery multiplication algorithm such that the low-cost and high-performance Montgomery modular multiplier can be implemented accordingly. The proposed multiplier receives and outputs the data with binary representation and uses only one-level carry-save adder (CSA) to avoid the carry propagation at each addition operation. This CSA is also used to perform operand pre computation and format conversion from the carry save format to the binary representation, leading to a short critical path delay at the expense of extra clock cycles for completing one modular multiplication. To overcome the weakness of extra clock cycle, a configurable CSA (CCSA), which could be one full-adder or two serial half-adders, is proposed to reduce the extra clock cycles for operand pre computation and format conversion by half. In addition, a mechanism that can detect and skip the unnecessary carry-save addition operations in the one-level CCSA architecture while maintaining the short critical path delay is developed. As a result, the extra clock cycles for operand pre computation and format conversion can be hidden and high throughput can be obtained. The experimental results show that the proposed Semi Carry Save Montgomery Modular Multiplier New (SCS-MM New) Architecture utilizes Delay of 2.5 ns, Clock cycle of 10, Area of 245 slices and achieves moderate Throughput of 387.78 Mbps.

Keywords: Carry Save Addition, Semi-carry save Adder, configurable CSA, Semi Carry Save Montgomery Modular Multiplier New (SCS-MM New)

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 $S = A \times B \times R^{-1} \pmod{N}$, 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.

C. McIvor et. al., [1] presents Modified Montgomery Multiplication and associated RSA modular exponentiation algorithms and circuit architectures, these modified multipliers use carry save adders (CSAs) to perform large word length additions. A key contribution has been to reformulate and solve the problem of modular multiplication as it relates to RSA exponentiation in such a way as to avoid this costly and previously necessary conversion addition. Thus, the lengthy and costly conventional additions required at each stage are avoided. Consequently, the critical path delay and hence the data throughput rate of the resulting Montgomery multiplier architectures are word length independent**.** Keklik Alptek˙in Bayam, Berna Ors [2] implemented that RSA cryptosystem was implemented on hardware, and then modified to be resistant against Differential Power Analysis attacks by using the Randomized Table Window method. Modular exponentiation is realized with Montgomery Modular Multiplication. The Montgomery modular multiplier has been realized with Carry-Save Add. Here they compare the protected implementation with the unprotected. Guilherme Perin, Daniel Gomes Mesquita, and Jo˜ao Baptista Martins [3] that the modular multiplication is

employed in modular exponentiation processes. The proposed systolic architecture presents a high-radix implementation with a one-dimensional array of Processing Elements. If the key size increases, the architecture can be easily modified by adding arithmetic cores, keeping the performance. João Carlos Néto, Alexandre Ferreira Tenca and Wilson Vicente Ruggiero [4] proposed a method to generate efficient implementations of sequential Montgomery Multiplication. The resulting hardware algorithm and architecture accelerate the modular multiplication by looking ahead the input data of two iterations and in some cases compressing two iterations in one, without increasing the iteration time too much.

This work presented a method to make a more efficient use of hardware resources required to compute the Montgomery Multiplication algorithm, optimizing the use of its adders when they are inactive due to some particular data values, by looking ahead the data to be processed on the adders we were able to compress two clock cycles in one cycle for some combinations of the input data. Philip Amberg Nathaniel Pinckney and David Money Harris [5] describes that the algorithm and design tradeoffs for multiple hardware implementations of parallel high-radix scalable Montgomery multipliers. Viktor Bunimov, Manfred Schimmler, Boris Tolg [6] designed modular multiplier to meet the predominant requirements of most modern devices, small chip area and low power consumption. The algorithm is superior to the original method by a factor of 2, with respect to both area and latency. It is well suited for a complexity efficient implementation with respect to both area and time. Neal Koblitz, Alfred Menezes, Scott Vanstone [7] are surveys the development of elliptic curve Cryptosystems. The problem of finding logarithms with respect to a generator in the multiplicative group of the integers modulo a prime, this idea can be extended to arbitrary groups and in particular, to elliptic curve groups. It provides relatively small block size, high speed and high security. Victor S. Miller [8] reviews the Montgomery multiplier that the general cryptographic protocols which are involved briefly describe the elliptic curves and review the possible attacks against such cryptosystem. R. L. Rivest, A. Shamir, and L. Adleman [9**]**, presents on momtgomery miultiplier is an encryption method is presented with the novel property that publicly revealing an encryption key does not thereby reveal the corresponding decryption. Vinodhin N, Suganya C [10] C they proposed pipelined VLSI architecture for RSA based on Montgomery modular multiplication, basis Modular multiplication forms a key operation in many public key cryptosystems. In the proposed architecture, maximum worst case delay is analyzed to enhance the throughput. In the path, additional buffers are introduced so that the clock is synchronized to reduce the worst case delay. As a result, pipelining concept is introduced which increases the speed and achieves a high throughput. A Akilavathi, A Vijaya Prabhu [11] they designed and implemented of Energy Efficient and High Throughput of Vedic Multiplier.

Designing a low power and high speed multiplier will have a large impact on applications like Image Processing, Convolution, Fast Fourier Transform, and Filtering and in microprocessors. The sutra named Urdhva-Triyagbhyam (Vertically and Cross wise) from the Ancient Indian Vedic Mathematics used in multiplication process since it has a unique way of calculations. The Vedic 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 of Montgomery multiplication shows that the produced output is 0.013mW hence in the proposed approach of Vedic multiplier the result will be enhanced to reduce the power and increase the speed and decrease the delay. Colin D. Walter Montgomery's [12] modular multiplication algorithm is commonly used in implementations of the RSA cryptosystem. Here they considered implementations of the RSA cryptosystem which use solely Montgomery's modular multiplication algorithm and shown that under standard, easily met, inexpensive conditions, the total encryption process never needs any extra subtractions to produce output in the correct range.

In this paper SCS-MM New VLSI Architecture was proposed, experimentation and analysis of the proposed multiplier was conducted using Xilinx software. The experimental results show that the proposed Semi Carry Save Montgomery Modular Multiplier New (SCS-MM New) Architecture utilizes Delay of 2.5 ns, Clock cycle of 10, Area of 245 slices and achieves moderate Throughput of 387.78 Mbps.

II. CONVENTIONAL MULTIPLIERS

A. SCS based Modular Multiplication-1

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 Algorithm 1. Note that the number of iterations in Figure .1.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 Algorithm 1. Figure 1 shows the architecture of SCS-based MM algorithm proposed in (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. 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 precomputed as D = B + N so that the computation of Ai × B + qi × N in step 4 of Algorithm 1 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.

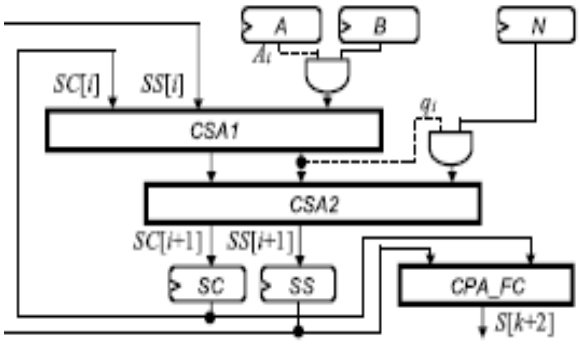

Figure 1: SCS-based Montgomery multiplication 1

Algorithm 1: SCS-based Montgomery multiplication-1

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];

B. SCS based Modular Multiplier -2

In SCS-MM-2 multiplier uses 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 Algorithm.1. 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. Figure 2.shows the architecture of the Montgomery multiplier (denoted as SCS-MM-2 multiplier). Note that the select signals of multiplexers M1 and M2 in Figure 2 generated by the control part are not shown in Figure 2 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.

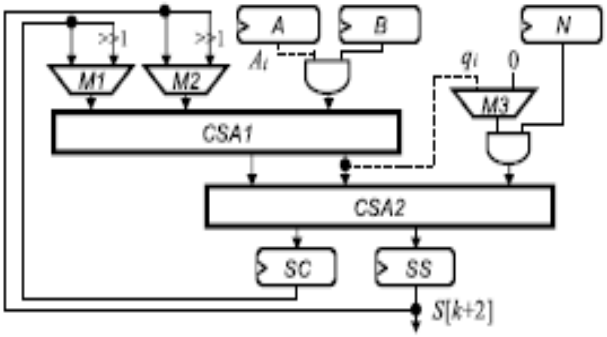

Figure 2: SCS-MM2

C. FCS based Modular Multiplier-1

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. [1] 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 2 and architecture of the FCS-MM-1 multiplier is shown in Figure 3 respectively. The barrel register full adder (BRFA) in Figure 3 consists of two shift registers for storing AS and AC, a full adder (FA), and a flip-flop (FF). On the other hand, the FCS-MM-2 multiplier proposed in [1] 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.

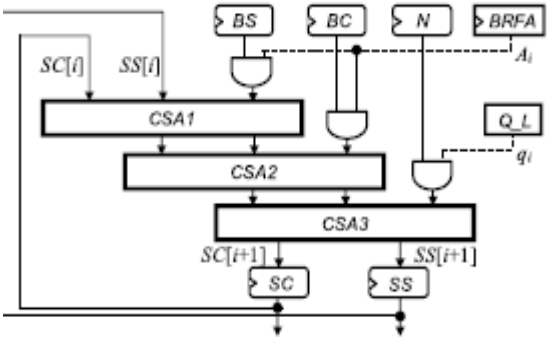

Figure 3: FCS mm1 multiplier

Algorithm 2: FCS mm1 Montgomery multiplier

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];

The modified SCS-based Montgomery MM algorithm is proposed 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.

The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM that is, we can pre-compute D = B + N and reuse the one-level CSA architecture to perform B+N and the format conversion. Figure 4 and Algorithm 3 shows the modified SCS-based Montgomery multiplication (MSCS-MM) algorithm and one possible hardware architecture, respectively. The Zero_D circuit in Algorithm 3 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 Algorithm 3. The carry propagation addition operations of B + N and the format conversion are performed by the one-level CSA architecture of the MSCS-MM multiplier through repeatedly executing the carry-save addition (SS, SC) = SS + SC + 0 until SC = 0. In addition, we also precompute Ai and qi in iteration i−1 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 Algorithm 3] 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.

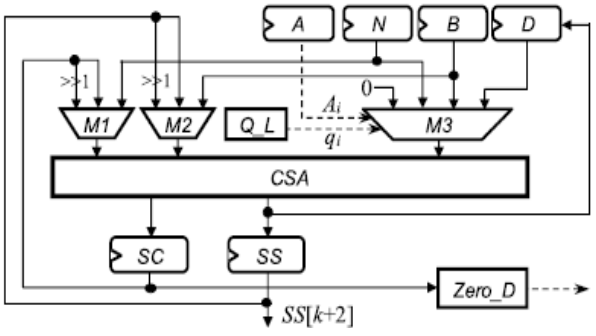

Figure 4: Modified SCS-based Montgomery multiplication Multiplier

Algorithm 3: Modified SCS-based Montgomery multiplication

Inputs: A, B, N (modules)
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. $q_i=(SS[i]_0 + SC[i]_0 + A_i \times B_0)\ mod\ 2;$
8. if($A_i = 0;\ q_i = 0$) x=0;
9. if($A_i = 0;\ q_i = 1$) x=N;
10. if($A_i = 1;\ q_i = 0$) x=B;
11. if($A_i = 1;\ q_i = 1$) 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];

III. PROPOSED MONTGOMERY MULTIPLICATION

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 Algorithm 4) using one-level CCSA architecture is proposed to significantly reduce the required clock cycles for completing one MM.

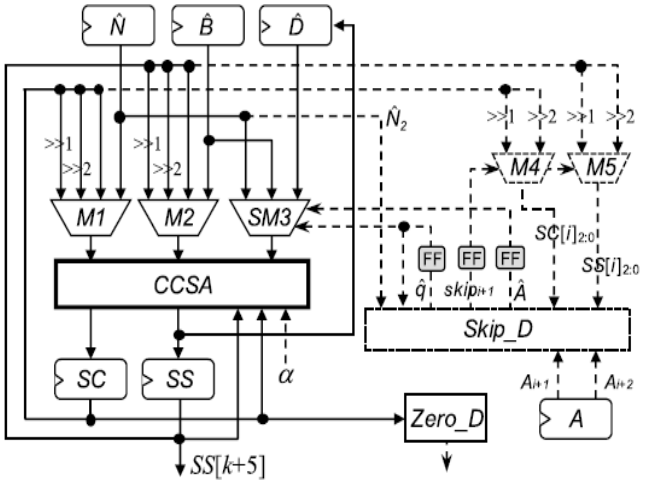

Figure 5: VLSI Architecture SCS-MM-New Multiplier

The hardware architecture of SCS-MM-New algorithm, denoted as SCS-MM-New multiplier, are shown in Figure 5 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 $skip_{i+1}$, $\hat{q}$, and $\hat{A}$ in the i^{th} iteration. Both M4 and M5 in Figure 5 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 Figure 1, the select signals of multiplexers M1 and M2 in Figure 5 are generated by the control part, which are not depicted for the sake of simplicity.

Algorithm 4: Proposed Semi Carry Save Montgomery Modular Multiplier New (SCS-MM New)

Input: A, B, $\widehat{N}$ $(new\ modules)$
Output: SS [k+5]

1. $\hat{B} = B \ll 3;\ \hat{q} = 0;\ \hat{A} = 0;\ skip_{i+1} = 0;$
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$\leq k+4$){
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), (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];

At the beginning of Montgomery multiplication, the FFs stored c are first reset to 0 as shown in step 1 of SCS-MM-New algorithm so that $\widehat{D} = \hat{B} + \hat{N}$ can be computed via the one-level CCSA architecture. When performing the while loop, the skip detector. In the next clock cycle of the i th iteration, SM3 outputs a proper x according to $\hat{q}$ and $\hat{A}$ generated in the i th 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 $skip_{i+1}$ = 0, SC_1 and SS_1 are selected. Otherwise, SC_2 and SS_2 are selected. That is, the right-shift 1-bit operations in steps 12 and 15 of SCS-MM-New algorithm are performed together in the next clock cycle of iteration i. In addition, M4 and M5 also select and output the correct SC $[i]_{2:0}$ and SS $[i]_{2:0}$ according to $skip_{i+1}$ 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 $\widehat{D} = \hat{B} + \hat{N}$ in steps 3 and 4. Finally, SS[k +5] in binary format is outputted when SC[k + 5] is equal to 0.

A. Internal Modules

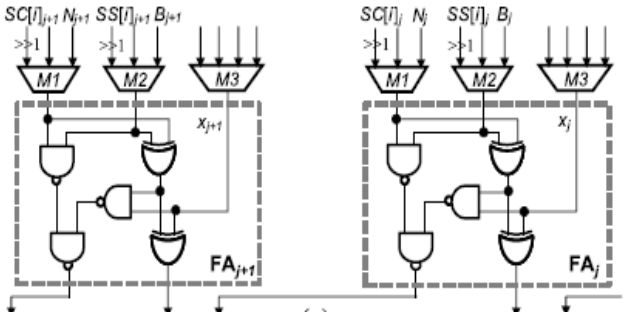

Figure 6(a): Conventional full adder circuit

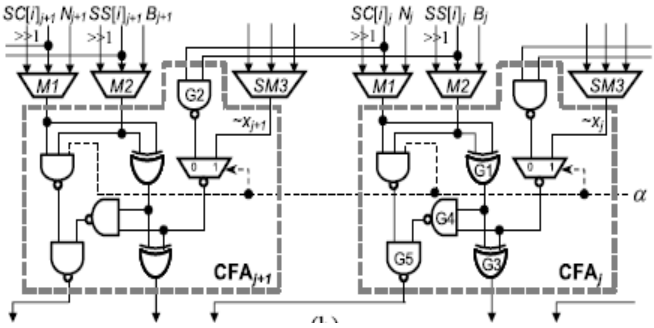

Figure 6(b): Proposed CFA circuit

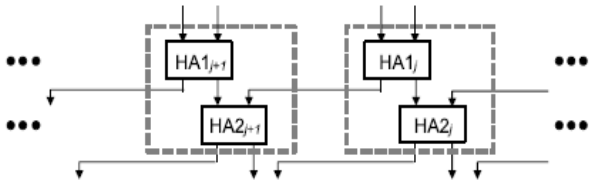

Figure 6(c): Two serial half adders

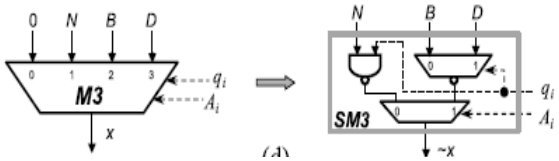

Figure 6(d): Simplified multiplexer SM3

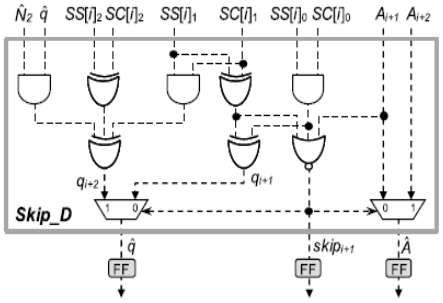

Figure 6(e): Skip _Detector

Figure 6(a) shows two cells of the one-level CSA architecture in Figure 4, each cell is one conventional FA which can perform the three-input carry-save addition. Figure 6(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 Figure 6(c). In this case, G1 of CFA*j* and G2 of CFA*j*+1 in Figure 6(b) will act as HA1 *j* in Figure 6(c), and G3, G4, and G5 of CFA*j* in Figure 6(b) will behave as HA2*j* in Figure 6(c).Moreover, we modify the 4-to-1 multiplexer*M3* in Figure 4 into a simplified multiplier *SM3* as shown in Figure 6(d) because one of its inputs is zero, where ~ denotes the INVERT operation. Note that *M3* has been replaced Skip_D show by *SM3* in the proposed one-level CCSA architecture shown in Figure 6(b). Therefore, the critical path delay of the proposed one-level CCSA architecture in Figure 6(b) is approximate to that of the one-level CSA architecture in Figure 6(a). As a result, steps 3 and 15 of Algorithm 3 can be replaced with (*SS*, *SC*) = 2H_CSA(*SS*, *SC*) and (*SS*[*k* + 2], *SC*[*k* + 2]) = 2*H_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 Algorithm 3 to further decrease the clock cycles for completing one Montgomery MM in Figure 6(e) is used to produce $skip_{i+1}$, $\hat{q}$, $\hat{A}$. 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 i^{th} iteration, and then selects the correct $\hat{q}\ and\ \hat{A}$ according to $skip_{i+1}$. At the end of the i th iteration,$\hat{q}$, $\hat{A}$ and $skip_{i+1}$ must be stored to FFs.

IV. RESULTS AND DISCUSSION

A. Simulation Result for Simplified Multiplexer SM3

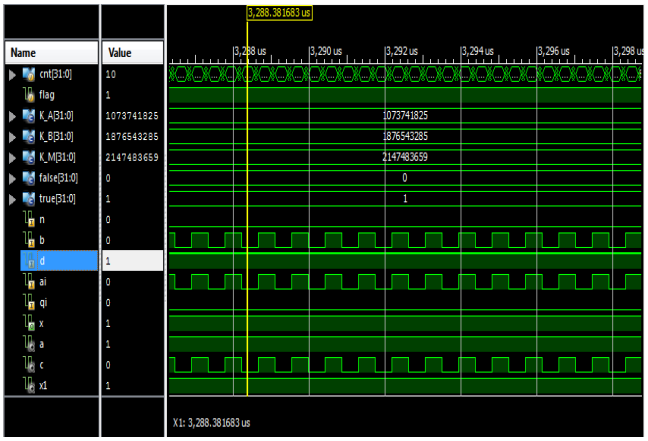

Figure 7: Simulation Result for Simplified Multiplexer SM3

The Figure 7 shows the modification of 4 to 1 multiplexer into Simplified Multiplexer SM3, because one of its inputs is zero as shown in Proposed Montgomery Modular Multiplication architecture. Where n, b, d, ai and qi are inputs and x is the output.

B. Simulation Result for Proposed Configurable Full Adder

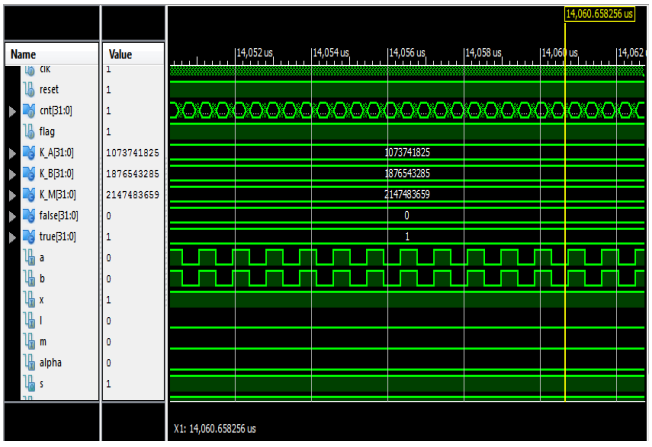

Figure 8: Simulation Result for Proposed Configurable Full Adder

In Figure 8, α=0, therefore CFA acts as two serial Half Adder circuit and perform two serial two-input carry save adder. It reduces the required clock cycle approximately a factor of two while maintaining a short critical path.

C. Simulation Result for Skip Detector

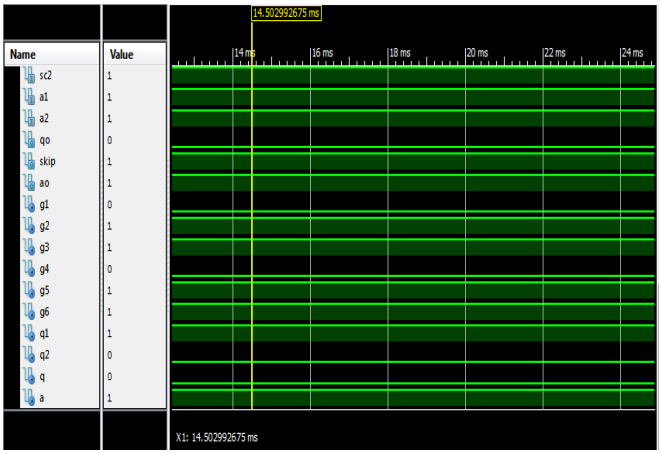

Figure 9: Simulation Result for Skip Detector

The Figure 9 shows Skip Detector first generates the qi+1, qi+2 and skipi+1 signal in ith iteration, and then select the correct q^ and A^ according to the skipi+1.

D. Simulation Result for Modular Iteration

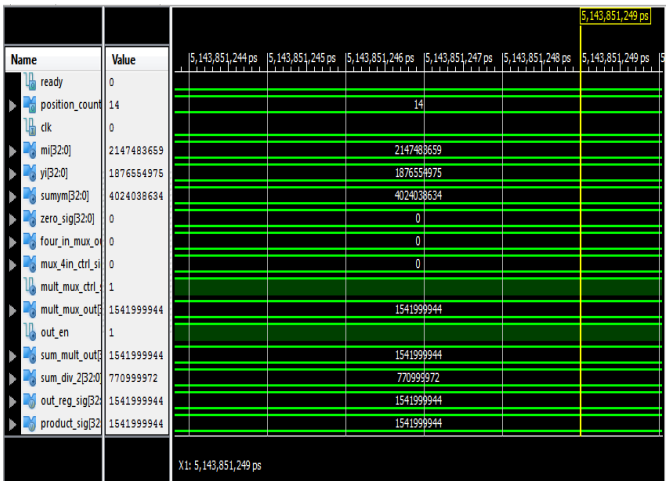

Figure 10: Simulation Result for Modular Iteration

The Figure 10 shows Simulation Result of Modular Iteration.

E. Simulation Result for Montgomery Modular Multiplication Top Model

The Figure 11 shows, simulation Result for Montgomery Modular Multiplication Top Model.

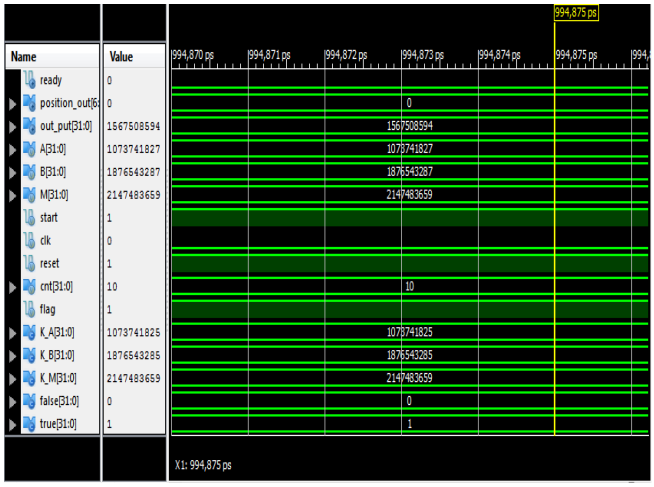

Figure 11: Simulation Result for Montgomery Modular Multiplication Top Model

This is Test Bench for the Montgomery modular multiplier with the 32 bit width, and it takes two numbers and modulus as the input and results the Montgomery product:

A*B*(R^{-1}) mod M

Where,

R^{-1} is the modular multiplicative inverse.

R*R^{-1} == 1 mod M

R = 2^word_length mod M

and word length is the binary width of the operated word (in this case 32 bit)

Preparation for test case:

A = 1073741826 in decimal

B = 1876543286 in decimal

M = 2147483659 in decimal

Result=mod(1073741826*1876543286*1659419191, 2147483659) = 1567508594

F. Analysis of SCS-MM New Multiplier for Delay and Area

TABLE I
Comparisons of different Montgomery Multipliers with 1024 Bit Key Sizes

Multi-pliers	SCS-MM-1	SCS-MM-2	FCS-MM-1	FCS-MM-2	FCS-MMM 42	SC-MM New
Cycles	1072	1049	1029	1029	822	10
Delay (ns)	4.93	5.60	5.80	6.01	5.56	2.5
Area (μm^2)	487171	406127	518214	677121	749076	245 Slices
Time (μs)	5.285	5.8744	5.9682	6.1843	4.5703	8.2519
Through-put(Mbps)	193.8	174.3	171.6	165.6	224.1	387.78

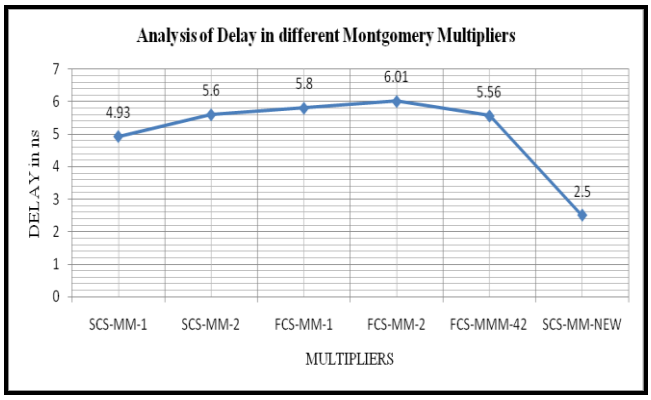

Figure 12: Comparison of Delay v/s Different MM Multipliers

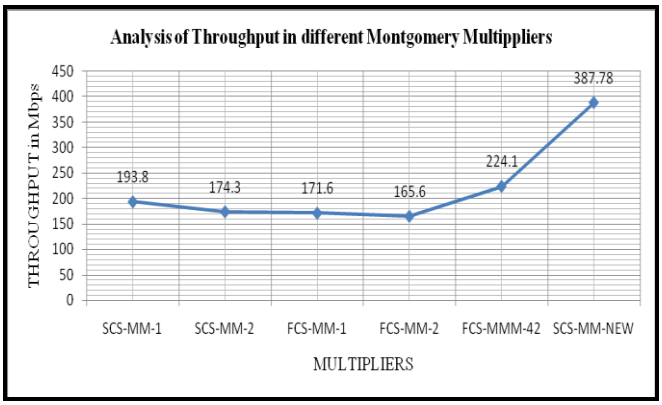

Figure 13: Comparison of Throughput v/s Different MM Multipliers

In SCS-MM-1 32 bit CPA for performing format conversion, which adds two 32 bit input and generates a 32 bit output at every cycle. Therefore the 32 bit CPA_FC will take 32 clock cycles to complete the format conversion of a 1024-bit SCS -MM. The extra CPA_FC enlarges the area and the critical path of the SCS-MM1 multiplier.

In SCS-MM2, for format conversion two-level CSA architecture is used so that CPA_FC can be removed. i.e. SS [k+2]=SS[k+2]+sc[k+2] in step 6 of Algorithm 1. It is replaced with the repeated carry-save operation (SS[k+2] , SC[k+2]) = SS[k+2]+SC[k+2] until SC[k+2] =0. However, the extra clock cycles required for 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.

FCS MM multiplier: to avoid the format conversion it maintains A, B, and S in the carry save representations (AS, AC), (BS, BC), (SS, SC), respectively. So it provides two FCS-MM methods denoted as FCS-MM1 and FCS-MM2, composed three-level and two-level

CSA architecture, respectively. It consists of the barrel register full adder (BRFA) for storing AS and AC, a full adder, and a flip-flop. FCS-MM2 needs two extra 4-1 multiplexer addressed by Ai and Qi and two more register to store DS and DC to reduce one level of CSA tree. Therefore the critical path delay of FCS-MM2 may be slightly reduced with a significant increase in hardware area compared with the FCS-MM1 multiplier.

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. The proposed Semi Carry Save Montgomery Modular Multiplier New (SCS-MM New) Architecture utilizes Delay of 2.5 ns, Clock cycle of 10, Area of 245 slices and achieves moderate Throughput of 387.78 Mbps.

V. REFERENCES

[1] C. McIvor, M. McLoone, and J. V. McCanny, "Modified Montgomery", IEE Proc.-Comput. Digit. Techn., Vol. 151, No. 6, Nov. 2004, pp. 402–408.

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

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

[4] J. C. Neto, A. F. Tenca, and W. V. Ruggiero, "A parallel k-partition method to perform Montgomery multiplication", Proc. IEEE Int Conf. Appl:-Specific Syst., Archit., Processors, Sep. 2011, pp. 251–254.

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

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

[7] N. Koblitz, "Elliptic curve cryptosystems", Math. Comput., Vol. 48, N. 177, 1987, pp. 203–20.

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

[9] R. L. Rivest, A. Shamir, and L. Adleman, "A method for obtaining digital signatures and public-key cryptosystems", Common.ACM, Vol. 21, No. 2, pp. 120–126, Feb. 1978.

[10] Vinodhin N, Suganya C, "Pipelined VLSI Architecture for RSA Based on Montgomery Modular Multiplication".

[11] A Akilavathi, A Vijaya Prabhu, "Design and Implementation of Energy Efficient and High Throughput Vedic Multiplier".

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