AN EFFICIENT MONTGOMERY MODULAR MULTIPLIER USING PARALLEL PREFIX ADDER

Sanduri Akshitha, Mrs.P.Navitha (Assistant Professor), Mrs.D.Mamatha (Assistant Professor)

Abstract

In modular arithmetic,Montgomery modular multiplication,more commonly reffered to as Montgomery multiplication, is a method for performing fast modular multiplication (MM). Modular multiplication is basic operation in public key cryptosystems like RSA.Montgomery modular multiplication being efficient is widely used.It is based on additions and shift operations.The proposed multiplier represents the data in binary representation and uses carry-save adder (CSA) to avoid the carry propagation at each addition operation.

We are proposing a Montgomery modular multiplication by using parallel prefix adders(PPA) which are a special case of carry look ahead adders.Parallel prefix adders reduce the carry propogation delay as compared to Ripple carry adders (RCA) and does not have the huge area overhead like Carry look ahead adder(CLA).

Index Terms— carry-save addition, Montgomery modular multiplier, public-key cryptosystem.

1. Introduction

In numerous open key cryptosystems, particular increase with expansive whole numbers is the most basic and slow activity. Along these lines, various calculations and equipment execution have been exhibited to complete the MM all the more rapidly,and Montgomery's calculation is a standout amongst the most understood MM calculations. Montgomery's calculation decides the remainder just relying upon the slightest huge digit of operands and replaces the muddled(confused) division in traditional MM with a progression of moving measured increases to deliver $S = A \times B \times R^{-1} \pmod N$, where N is the k-bit

modulus, R^{-1} is the backwards of R modulo N, and $R = 2^k \bmod N$ .Therefore, it can be effectively executed into VLSI circuits to accelerate the encryption/unscrambling process. In any case, the three-operand expansion in the cycle circle of Montgomery's calculation as appeared in stage 4 of Fig. 1 requires long carry propogation for large operands in binary representation.To solve this issue,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 approches can be divided into semi-carry save (SCS) and full carry-save(FCS) formats.

In semi-carry save format, the input and output operands (i.e.,A,B,N and S) of the Montgomery MM are reprsented in binary,but intermediate results of shifting modular additions are in carry-save format to avoid the carry propogation.the format conversion from the carry-save format of the final modular product into its binary representation is needed at the end of each modular multiplication.The full carry-save format 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.

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

2. Modular Multiplication Algorithm

Fig. 2 demonstrates the radix-2 rendition of the Montgomery.MM calculation (indicated as MM calculation). As specified before, the Montgomery particular item S of An and B can be acquired as S = A × B × R−1 (mod N), where R−1 is the opposite of R modulo N. That is, R × R−1 = 1 (mod N). Note that, the documentation Xi in Fig. 1 demonstrates the I th bit of X in double portrayal. Furthermore, the documentation Xi: j demonstrates a portion of X from the I th bit to j th bit.

2.1.SCS-BasedMontogmery 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.

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.

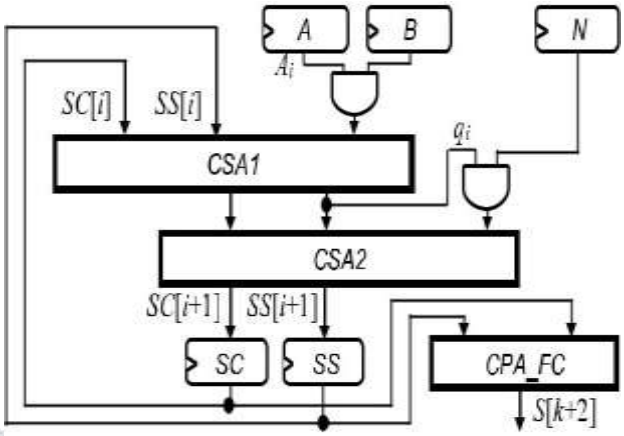

Fig. 3. SCS-MM-1 multiplier.

Fig: 3 shows the architecture of SCS-based MM algorithm which consists of one two-level CSA architecture and one format converter.

2.2. 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*).FCS-MM and multiplier, made out of one five-to-two (three-level) and one four-to (two-level) CSA design, individually. The calculation and design of the FCS-MM multiplier are appeared in Figs. 4 and 5, separately.

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. 4. FCS-MM-1 Montgomery multiplication algorithm.

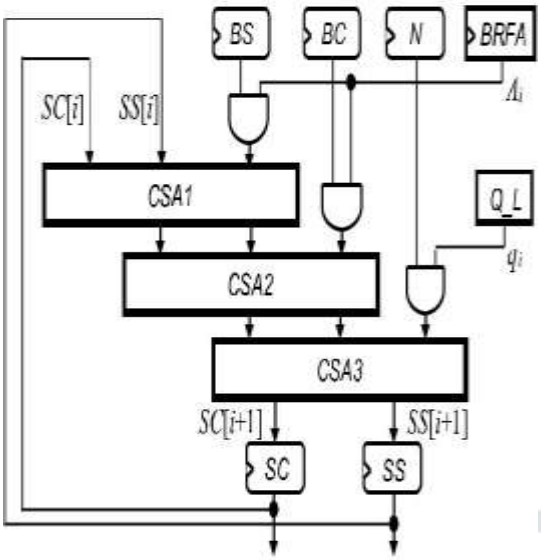

Fig. 5 FCS-MM-1 multiplier.

3. Proposed Montgomery Multiplication

In this,we proposeMontgomery 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

3.1 Carry save adder

A carry-save adder is a type of digital adder, used in computer microarchitecture to compute the sum of three or more *n*-bit numbers in binary. It differs from other digital adders in that it outputs two numbers of the same dimensions as the inputs, one which is a sequence of partial sum bits and another which is a sequence of carry bits. Montgomery multiplication, which depends on the rightmost digit of the result, is one solution; though rather like carry-save addition itself, it carries a fixed overhead, so that a sequence of Montgomery multiplications saves time but a single one does not. Fortunately exponentiation, which is effectively a sequence of multiplications, is the most common operation in public-key cryptography.

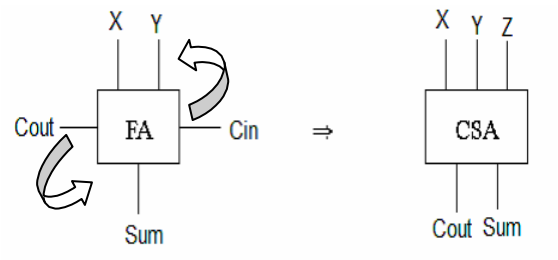

Figure 6: The 1-bit carry save adder block is the same circuit as a full adder.

3.2 Multilevel Carry-Save Adder

Multilevel CSA use a number of CSAs interconnected as a multilevel adder tree to add more than one number per cycle. The number of stages of CSA algorithm decides the basic cycle time of the addition process.

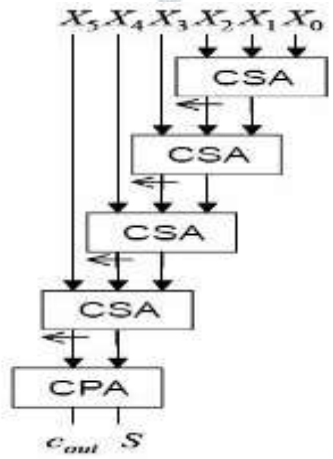

Figure 7 CSA tree

3.3 Critical Path Delay Reduction

The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM 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. Fig. 8 and 9 shows the modified SCS-based Montgomery multiplication (MSCS-MM) algorithm and one possible hardware architecture, respectively. The *Zero_D* circuit in Fig.8 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. 9.

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 (*S S*, *SC*) = *SS* + *SC* + 0 until *SC* = 0. In addition, we also pre compute *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 $T_{MUX4} + T_{FA}$.

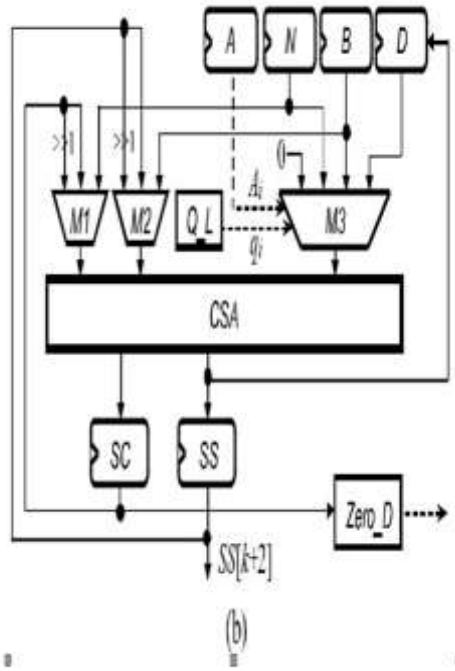

Figure 8 MSCS-MM multiplier.

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

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

1. $(SS, SC) = (B + N + 0)$;
2. while $(SC != 0)$
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 $SS[k+2]$;

Fig.9 Modified SCS-based Montgomery multiplication algorithm.

However, in addition to performing the three-input carry-save additions [i.e., step 12 of Fig 9] $k + 2$ 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*.

That is, ∼3*k* 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.

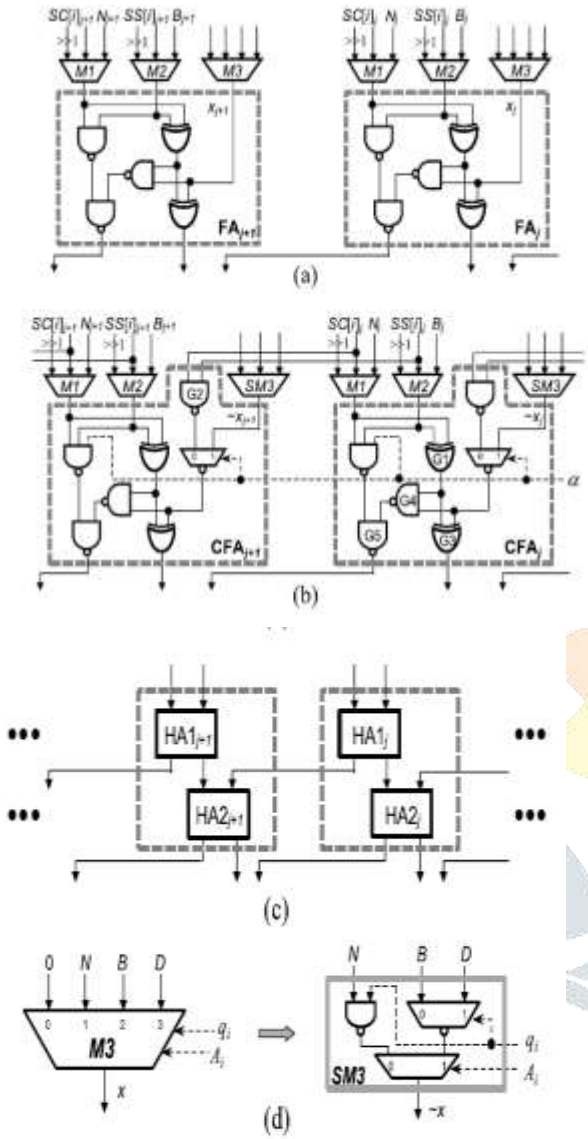

Fig. 10 (a) Conventional FA circuit. (b) Proposed CFA circuit. (c) Two serial HAs. (d) Simplified multiplexer *SM3*.

In addition, we also skip the unnecessary operations in the for loop (steps 6 to 13) of Fig. 9 to further decrease the clock cycles for completing one Montgomery MM. The crucial computation in the for loop of Fig.9 is performing the following three-to-two carry-save addition:

$(S S[i + 1], SC[i + 1]) = (S S[i] + SC[i] + x)/2$

where the variable x may be 0, N , B, or D depending on the values of A_i and q_i. Accordingly, the signal $skip_{i+1}$ used in the i th iteration to indicate whether the carry-save addition in the $(i + 1)$ th iteration will be skipped can be expressed as

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

where ∨ represents the OR operation. If $skip_{i+1}$ generated in the i th 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 i th 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., $skip_{i+1} = 1$), $SS[i + 1]$ and $SC[i + 1]$ produced in the i th iteration must be right shifted by two bit positions and the next clock cycle will go to iteration $i + 2$ to skip the carry-save addition of the $(i + 1)$th iteration. In this situation, not only $qi+1$ and $Ai+1$ but also $qi+2$ and $Ai+2$ must be produced and stored to FFs in the i th 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 $(qi+1, Ai+1)$ or $(qi+2, Ai+2)$ according to the $skipi+1$ signal produced in the i th iteration. That is, $(\hat{}\, q, \hat{}A) = (qi+2, Ai+2)$ if $skipi+1 = 1$. Otherwise, $(\hat{}\, q, \hat{}A) = (qi+1, Ai+1)$.

3.4 Proposed Algorithm and Hardware Architecture

On the bases of critical path delay reduction, a new SCS-based Montgomery MM algorithm (i.e., SCS-MM-New algorithm shown in Fig. 12) 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 the i th 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 $skipi+1 = 1$. In addition, the ending number of

iterations in SCS-MM-New algorithm is changed to k + 4 instead of k + 1 in Fig. 11. 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 qi+1, qi+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. 12, 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.

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. 11. SCS-MM-New algorithm.

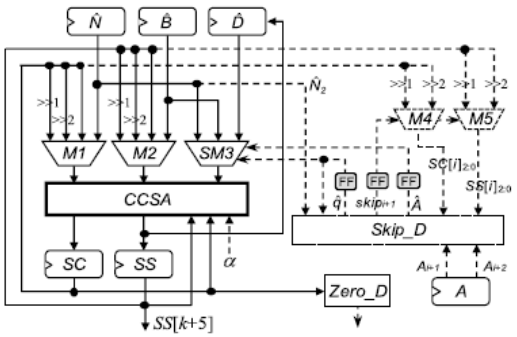

Fig. 12. SCS-MM-New multiplier.

Skip_D is developed to generate skip$i$+1, $\hat{}q$, and $\hat{}A$ in the i th iteration. Both *M4* and *M5* in Fig. 12 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.The select signals of multiplexers *M1* and *M2* in Fig. 12 are generated by the control part, which are not depicted for the sake of simplicity.

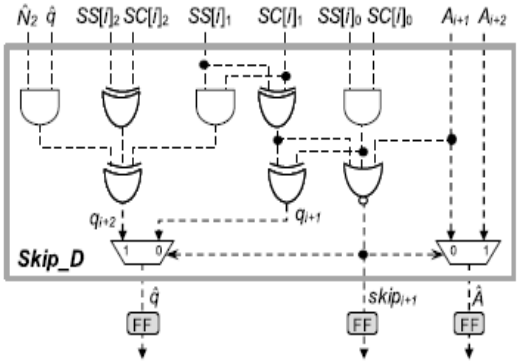

Fig. 13. Skip detector *Skip_D*.

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 qi+1, qi+2, and skip$i$+1 signal in the i th 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 i th iteration, $\hat{}q$, $\hat{}A$, and skip$i$+1 must be stored to FFs. 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 i th 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 i th 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, ˆ *q* and ˆ*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 ˆ*D* = ˆ*B* + ˆ*N* in steps 3 and 4. Finally, *SS*[*k* +5] in binary format is outputted when *SC*[*k* + 5] is equal to 0 .

4.EXTENSION METHOD

4.1 Parallel Prefix Adder

In proposed system, a parallel prefix adder(**PPA**) is used to reduce carry propogation delay.koggestone adder is one type of parallel prefix adder. VLSI Integer adders find applications in Arithmetic and Logic Units (ALUs), microprocessors and memory addressing units. Speed of the adder often decides the minimum clock cycle time in a microprocessor. The need for a Parallel Prefix Adder (PPA) is that it is primarily fast when compared with a ripple carry adder. PPA is afamily of adders derived from the commonly known carry look ahead adders. These adders are suited for additions with wider word lengths. PPA circuits use a tree network to reduce the latency to 2(log) n,where 'n' represents the number of bits..The Kogge–Stone adder is a parallel prefix form carry look-ahead adder. Other parallel prefix adders include the Brent-Kung adder, the Han Carlson adder, and the fastest known variation, the Lynch-Swartzlander Spanning Tree adder.PPA pre-computes and generates the signals,it involves less delay .

4.2 Kogge – Stone Adder(KSA)

KSA is a parallel prefix form carry look ahead adder, it is widely considered as the fastest adder and is widely used in the industry for high performance arithmetic circuits

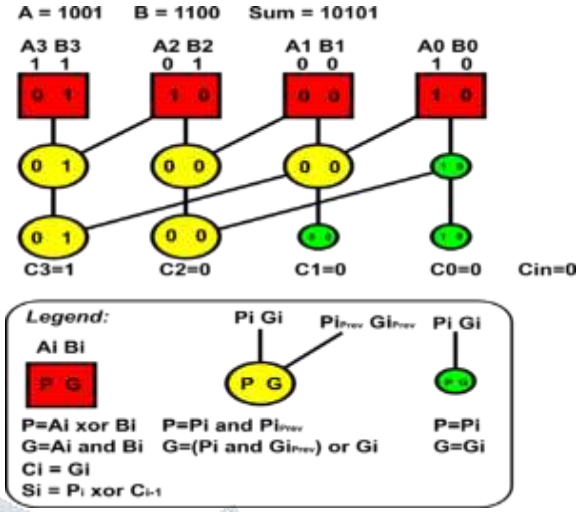

Figure 14 .kogge stone adder example

The Kogge–Stone adder takes more area to implement than the Brent–Kung adder, but has a lower fan-out at each stage, which increases performance for typical CMOS process nodes. However, wiring congestion is often a problem for Kogge–Stone adders. The Lynch-Swartzlander design is smaller, has lower fan-out, and does not suffer from wiring congestion; however to be used the process node must support Manchester Carry Chain implementations. The general problem of optimizing parallel prefix adders is identical to the variable block size, multi level, carry-skip adder optimization problem, a solution of which is found in an example of a 4-bit Kogge–Stone adder is shown in the diagram. Each vertical stage produces a "propagate" and a "generate" bit, as shown. The culminating generate bits (the carries) are produced in the last stage (vertically), and these bits are XOR'd with the initial propagate after the input (the red boxes) to produce the sum bits. E.g., the first (least-significant) sum bit is calculated by XORingthe propagate in the farthest-right red box (a "1") with the carry-in (a "0"), producing a "1". The second bit is calculated by XORingthe propagate in second box from the right (a "0") with C0 (a "0"), producing a "0".

5.Simulation And Synthesis Results:

5.1 Synthesis results:

Synthesis results of parallel prefix adder is shown in figure 5.1.Here a, b are inputs ,sum is output and i is 32 bit integer.

Figure 5.1 synthesis results of parallel prefix adder

5.2 simulation results

Simulation results describe the design summary. It contains number of flip-flop, LUT's, IOBs, timing delays are used.

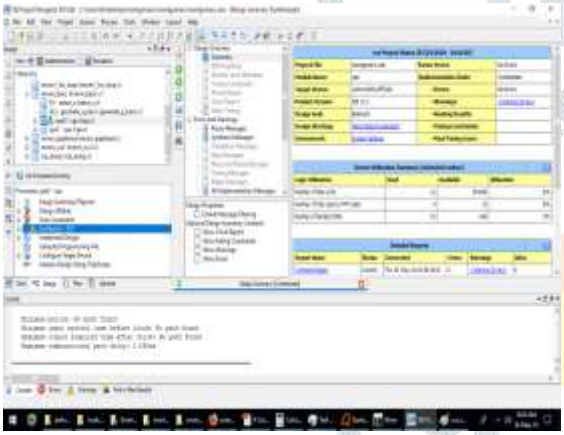

Figure 5.2 design summary of carry propagate adder

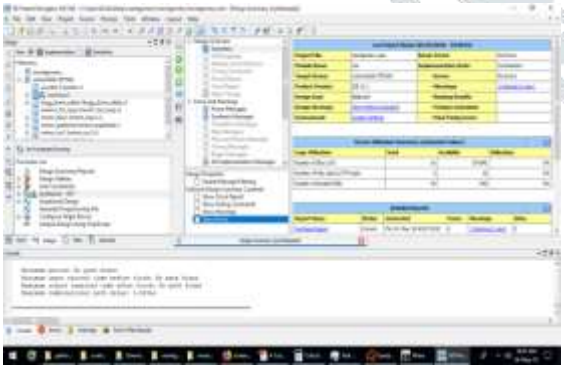

Figure 5.3 design summary of carry save adder

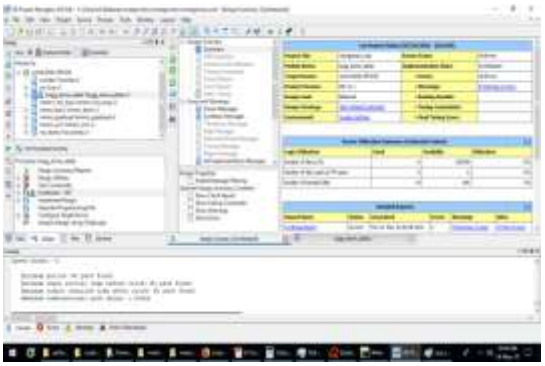

Figure5.4 design summary of kogg stone adder

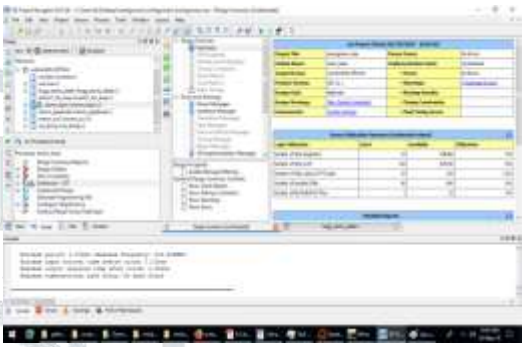

Figure 5.5 design summary of Montgomery modular multiplication

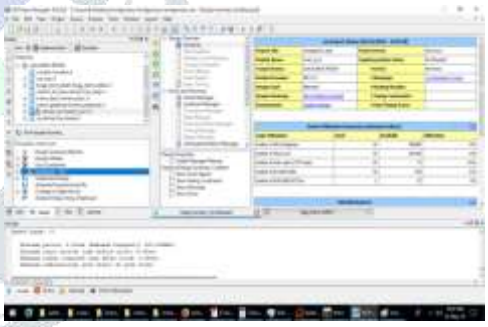

Figure 5.6 design summary of Montgomery modular multiplication of semi carry save adder

6. CONCLUSION

In this we have analysed an efficient Montgomery Modular multiplier using a parallel prefix adder.A parallel prefix adder reduces the delay of the system which further increases the speed of the system. Here we have used a Kogge-stone adder which is one type of parallel prefix adder , in which carries are generated fast by computing them in parallel at the cost of increased area.

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 cryptosys-tems," *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.

Author profile:

Sanduri Akshitha,she received bachelors of degree in 2015 from Electronics and Communication of engineering from Sudheer Reddy college of engineerng andctechnoogy for women.She is pursuing M.Tech in VLSI System Design from CMR Institute of Technology.

Mrs.P.Navitha

She is working as Assistant professor in CMR Institute of Technology and has 6 years experience in teaching field.

Mrs.D.Mamatha

She is working as Assistantprofessor in CMR institute of Technology nd has 4 years experience in teaching field