Low-Delay High-Performance VLSI Architecture using Montgomery Modular Multiplication

R. Ponnagan[1]
Assistant Professor, Dept. of ECE
K.Ramakrishnan College of Technology
Samayapuram, Trichy.

C. Archana[2], R. Bhuvaneswari[3], R. Boovitha[4]
Department of ECE
K.Ramakrishnan College of Technology
Samayapuram, Trichy.

***Abstract*—There is more advancement in VLSI technology for designing combinational and sequential circuits. Among various techniques we have chosen a CMOS (Complementary Metal Oxide Semiconductor) and Verilog coding technique to design the combinational circuits such as full adder and ripple carry adder. Full adder is the basic combinational circuit which performs arithmetic operations. Multiple full adder circuits can be cascaded to add an N-bit number i.e. Ripple carry adder. This paper proposes a simple and an efficient modular multiplication using a Carry Skip Adder. The proposed algorithm improves the efficiency of the multiplication. We made power analysis comparisons by using several types of adders and found that Carry skip adder consumes less power than other adders. The simulation result shows that the proposed combinational circuits have better performance and it maintains low complexity of the design.**

***Index Terms*— Carry-Skip addition, low-power architecture, Montgomery modular multiplier.**

I. INTRODUCTION

IN MANY systems, large integers in modular multiplication (MM) is the most critical and time-consuming operation. Therefore, many algorithms have been presented to carry out the modular multiplication more fastly, among that Montgomery's algorithm is one of the efficient MM algorithm. Depending on the least significant digit of the operands, Montgomery's algorithm determines the quotient and replaces the complex division in terms of conventional modular multiplication with a series of shifting modular additions to produce the result as $S = A \times B \times R^{-1} \pmod{N}$, where N is the K-bit modulus, R^{-1} is the inverse of R modulo N and $R=2^K \bmod N$. In order to speed up the encryption/decryption process, it is implemented into the Very Large Scale Integration (VLSI) circuits. Let us see the Montgomery's algorithm by using three operand for addition. Fig.1 represents, for large operands in binary representation requires long carry propagation.

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

Fig. 1 MM algorithm

II.MODULAR MULTIPLICATIONALGORITHMS

A. Montgomery Multiplication

Montgomery multiplication, is a method for performing fast modular multiplication. However, when many products are required, as in modular exponentiation , the conversion to Montgomery form is a negligible fraction of the time of the computation, and performing the computation by Montgomery multiplication is faster than the available alternatives. Many important cryptosystems such as RSA exchange is based on arithmetic operations modulo a large number, and for these cryptosystems, the increased speed afforded by Montgomery multiplication can be important in practice. Fig. 1 shows the Montgomery MM algorithm (denoted as MM algorithm) for radix-2 version. As mentioned earlier, $S = A \times B \times R{-}1 \pmod{N}$ represents the Montgomery modular product S of A and B, where R^{-1} is the inverse R modulo N. where, $R \times R{-}1 = 1 \pmod{N}$. Note that, the notation Ai in Fig. 1 shows the i^{th} bit of A in binary representation. In addition, the notation $Ai{:}\ j$ indicates a segment of A from the i^{th} to j^{th} bit. S's convergence range in MM algorithm is $0 \le S<2N$, $S = S - N$ is an additional operation is required, if $S \ge N$ which remove the oversize residue. step 6 is to eliminate the final comparison and subtraction which represented in Fig. 1.

Walter [2] changed the value of R to $k+2$ and $2^{k+2} \bmod N$ and the number of iterations , respectively. Nevertheless, the very large operand addition still restricts the performance of MM algorithm for long carry propagation. , the intermediate result S of shifting modular addition kept in the carry-save presentation (SS, SC), in order to avoid the long carry propagation. To remove the final comparison and subtraction [2] the number of iterations been changed from k to $k+2$. The format conversion from the carry-save format into its binary format is needed. Therefore, the 32-bit will take 32 clock cycles to complete the format con-version of a 1024-bit in based Montgomery multiplication. The extra clock cycles enlarges the critical path and area of the

multiplier. The works in [6] and [7] states that the computation of $A_i \times B + q_i \times N$ which is pre computed $D=B+N$ can be simplified into one selection operation. . One of the operands will be chosen among 0, *N*, *B*, and *D* if (A_i , q_i)=(0, 0), (0, 1), (1, 0), and (1, 1), respectively. In this multiplier, only one-level CSA architecture is required to perform the carry-skip at addition the expense of one extra 4-to-1 multiplexer. Operand *D* value can be store in an additional register.

III. CARRY SAVEADDER

In many cases we need to add several operands together, carry save adders are ideal for this type of addition. A carry save adder consists of a ladder of stand-alone full adders, which carries out a number of partial additions in every steps. The principal idea is that the carry has a higher power of 2 and thus is routed to the next column. Doing additions with Carry save adder saves time and logic.

In this method, for the first 3 numbers a row of full adders are used in Fig. 3.4. Then a row of full adders is added for each additional number. The final results, SUM and CARRY in the form of two numbers, are then summed up with a carry propagate adder or any other adder.

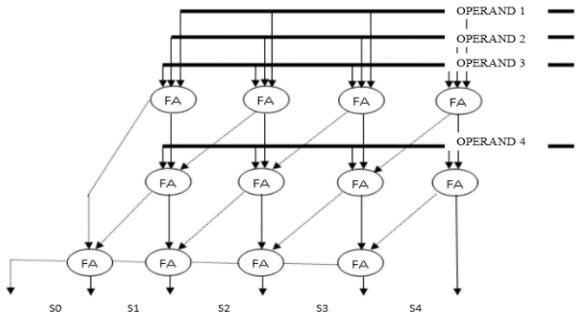


Fig 3.1 Carry Save Adder for 4 Bit Operand

When n bit adder is implemented, still carry is move to the next stage, of same row, even though inputs to the lower next stage is ready, it kept in wait state, until the carry and sum output didn't come from the above stage. This induces delay, now it can be optimized, if carry out is passed diagonally to the lower next stage, instead of rippling to the next stage of same row. Carry computation is not performed, but it is saved up to last row, in the last bottom row, carry is rippled however, where the results are obtained finally, but it significantly reduce the amount of delay occurred due to rippling operation.

IV.PROPOSED MONTGOMERY MULTIPLICATION

In this section, we propose a Montgomery MM algorithm for carry skip adder to reduce the critical path delay of Montgomery multiplier. In addition, the disadvantage of more clock cycles for completing one multiplication can be rectified while maintaining the advantages of short critical path delay and low hardware complexity. The time delay can also be reduced while perform the multiplication operation on the carry skip adder. Carry skip adder plays a vital role among all other adders in the area of power, time and area.

CARRY-SKIP ADDER

In binary system, carry is either can be 0 or 1.only two possibility of value provides feasibility for choosing between two. So if there is a mechanism to select carry, or better say, skipping carry through several stages as shown in Fig. 4.1, then delay minimization can be better obtained. In the carry skip adder to speed-up operation, carry propagation is skipped to position i. A carry-skip adder reduces the carry-propagation time by skipping over groups of consecutive adder stages.

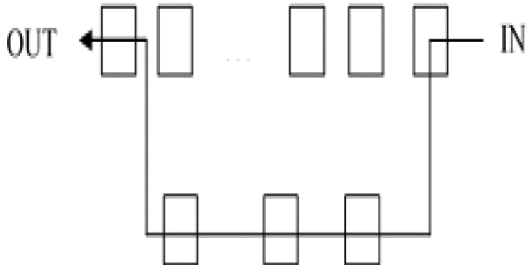


Fig. 4.1 Formal working of Carry Skip Adder

The carry-skip adder is usually speed when compare to the carry look-ahead technique, but it requires less chip area and consumes less power. To implement carry-skip adder, stages are divided into r–bit blocks of simple carry scheme. In each block, a ripple carry adder is utilized to produce the sum and carry for each block. Every block generates a block propagate and block generate signal. Also for the given column, CSKA uses the carry out equation in terms of the carry in signal.

$C_{k+1} = G_k + P_k . C_k$ ….carry out of i^{th} stage

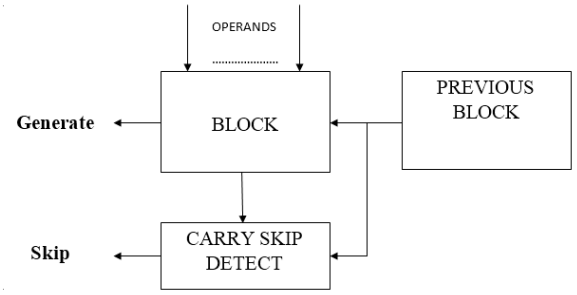
Also Generate and Propagate signals used by the carry skip

$G_k = A_k . B_k$

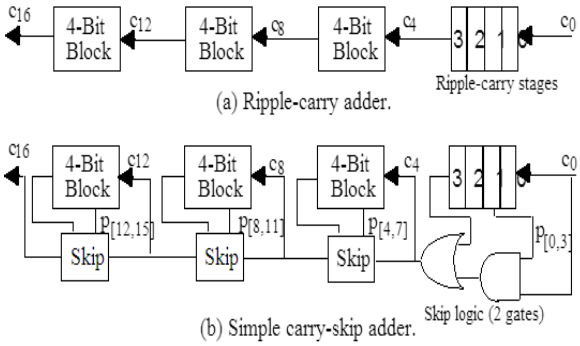
$P_k = A_k \oplus B_k$ ….carry propagate of i^{th} stage

From this equation, it can be seen that setting the carry-in signal to zero causes the carry out to serve as a block generate signal. Therefore, an *N* bit *AND* gate is also used to form block propagate signal. The block generates and propagates signals produce the input carry to the next block. To implement carry skip adder, stages are divided into blocks

Carry skip adder reduces the time needed to propagate carry by skipping over groups of consecutive adder stages. It requires less chip area and consumes less power than other adders. It has an execution time proportional to $\sqrt{n}$.



SIMPLE CARRY-SKIP ADDERS



SOME IMPORTANT INTERPRETATION ABOUT CARRY:-

From the full adder truth table we can easily interpret, carry out is killed or deleted when both of the inputs to the full adder are zero, irrespective of carry in whether it is 1 or 0. The 2nd observation is carry which is feed by the previous stage propagates to the next stage when either of the inputs to the full adder at present stage is 1(high).The another observation states carry is generated at present stage and will be feed to next stage when both of the inputs to the full adders are high. Hence in the expression $c_{out}=a_i.b_i+(a_i+b_i).c_i$, 1st term tells about carry generation at the present stage, and 2nd term states about carry propagation to the next stage(look we used XOR gate there in the design of full adder, in place of or gate in 2nd term here, which gives is the original carryout expression, just to avoid use of extra OR gate in design, as it does effect either XOR gate or OR gate is used for carry out since the only difference caused between 2 gates is for when both of the inputs are high ,that even is compensated, because we consider carry generation by 1st term in above expression to rise carry out high.

A carry-skip adder in Fig. 4.2, reduces the carry-propagation time by skipping over groups of consecutive adder stages. The carry-skip adder is usually speed to the carry look-ahead technique, but it requires less chip area and consumes less power.

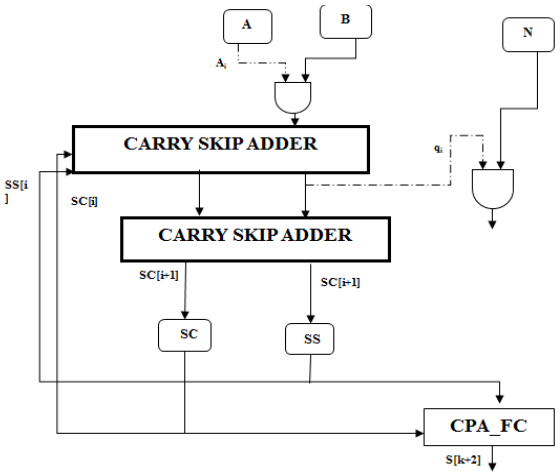


Fig.4.2 MM-1 Multiplier

In the carry-skip adder, any adder stage can be skipped for which Pm = A xor B = 1, where Pm indicates the m^{th} carry propagate. The adder structure is divided into blocks of consecutive stages with a simple ripple-carry scheme. Every block also generates a carry-propagate signal that equals 1 if all stages internal to the block satisfy Pm = 1. This signal can be used to allow an incoming carry to skip all the stages within the block and generate a block-carry-out. Fig. 8 shows an example block consisting of k bit positions j, j+1... j+k-1.

V.OUTPUT AND ITS DESCRIPTION

In this section, we first analysis the delay and power of the proposed carry skip adder. Here the delay and power are compared with the different adders. In addition, we simulate the adder and provide the result for the modular multiplication. Finally several Montgomery multipliers are implemented and synthesized to demonstrate the efficiency of the proposed approach.

A.SIMULATION OF SCS-MM MULTIPLIER

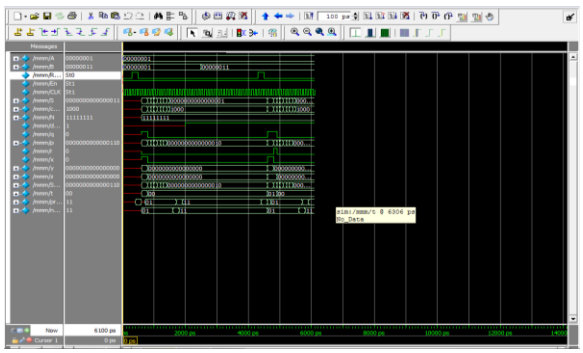


Fig. 5.1 Proposed system i.e. having two carry skip adder

The above Fig.5.1 show that Proposed system i.e. having two carry skip adder which perform repetitive addition in order to get a multiplication result by using Montgomery Modular Multiplication.

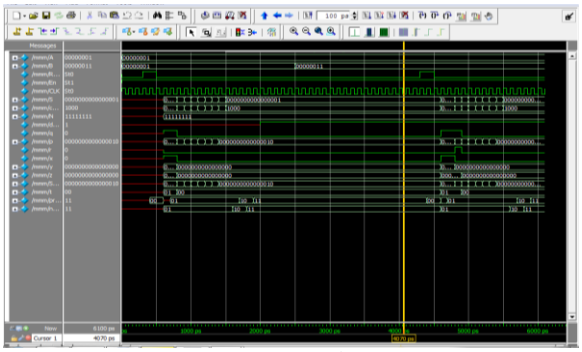


Fig. 5.2 Overall simulation of MMM algorithm

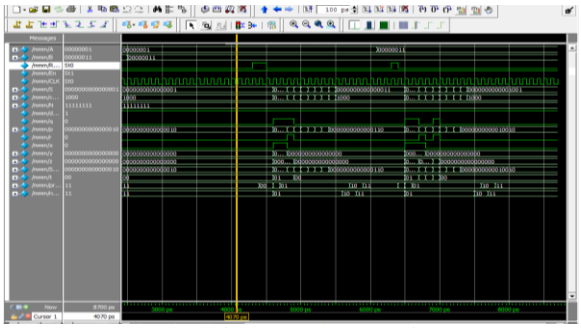


Fig. 5.3 Continuation of Overall simulation of MMM algorithm

B.POWER AND TIMING ANALYSIS

The Time Quest analyzer uses industry-standard Synopsys Design Constraint (SDC) methodology for constraining designs and reporting results.. Shown in fig. 5.4.

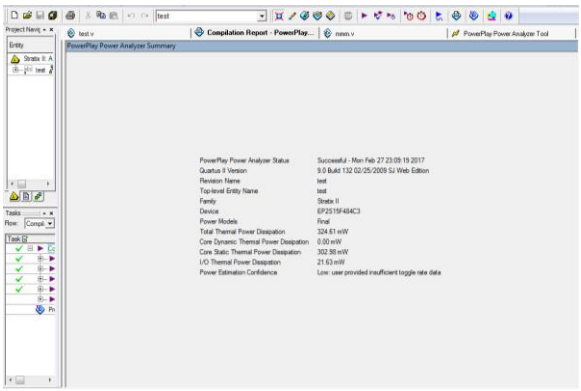


Fig. 5.4 Power analysis of SCS-MM multiplier

The Power Play Power Analyzer performs post fitting power analysis and produces a power report that highlights, by block type and entity, the power consumed. Shown in fig. 5.5

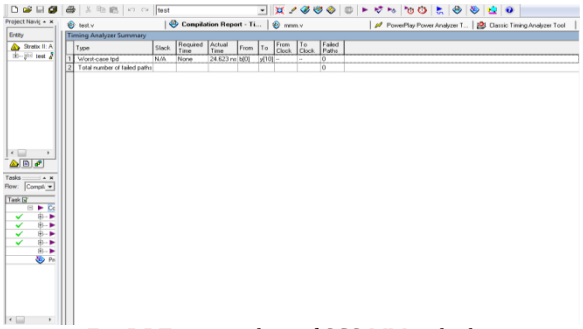


Fig. 5.5 Timing analysis of SCS-MM multiplier

These are the power and timing analysis of SCS based Montgomery modular multiplication by using carry skip adder.

VI.CONCLUSION

CSA based Montgomery modular multiplication maintain a format leading to fewer clock cycle and also had a large delay. To improve the performance of the Montgomery modular multiplication, this paper has been modified the low delay high performance Montgomery modular multiplier. The proposed multiplier used carry skip adder which skipped the unnecessary addition operation. Finally the experimental result showed that the proposed approaches are indeed capable of enhancing the performance and reducing the delay of the Montgomery modular multiplier maintaining low hardware complexity.

• ADDER	• DELAY
• MMM algorithm with **2** carry skip adder in a block	• 24.623ns
• MMM algorithm with **1** carry save adder in a block	• 18.304ns

REFERENCE

[1] R. L. Rivest, A. Shamir and L. Adleman “A method for obtaining digital signatures and public key cryptosystems”, Published in 2010.

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

[3] T. Dhinesh Kumar & S. K. Srivatsa “An effectiveMontgomery algorithm using multiplier circuits”, Published in 2011.

[4] M. Navya, A. Sireesha, B. Ramanjaneyulu “High performance Montgomery modular multipliers for RSA cryptosystem”, Published in 2014.

[5] S. Ashwini, P. Thirapaiah“A high- speed Montgomery modular multiplication algorithm to reduce the energy consumption based on RSA cryptosystems”, Published in 2015.

[6] A. Akilavathi, A. VijayaPrabu “Design and implementation of energy efficient and high throughput vedic multiplier”, Published in 2016.

[7] Aleixandre F. Tenca, Member, IEEE, and Cetin K. Koc¸ Senior Member, IEEE “A Scalable Architecture for Modular Multiplication Based on Montgomery’s Algorithm”, Published in 2003.

[8] Yuan-Yang Zhang, Zheng Li, Lei Yang, Shao-Wu Zhang “An efficient CSA architecture for Montgomery modular multiplication”, Published in 2007.

[9] Jun Han, Shuai Wang, Wei Huang, Zhiyi Yu “Parallelization of Radix-2 Montgomery Multiplication on Multicore Platform”, Published in 2013.