IOSR Journal of Electronics and Communication Engineering (IOSR-JECE)
e-ISSN: 2278-2834,p-ISSN: 2278-8735
PP 28-33
www.iosrjournals.org

Design Of High Performance Scs-Based Montgomery Multiplier

S.Ramya, S.Sangeetha, A.Veena BakkyaLakshmi, B.Zeenathul Asma, Mr.P.Muralikrishnan

Dept of Electronics and Communication Engineering K.Ramakrishnan College of Engineering Samayapuram , Trichy.
kuttynandhinibakkya@gmail.com
Assistant Professor Dept of Electronics and Communication Engineering K.Ramakrishnan college of Engineering Samapuram, Trichy.

Abstract: *A simple and efficient multiplication algorithm for the low-cost and high-performance Montgomery modular multiplier can be implemented accordingly. The proposed multiplier uses one-level carry-save adder (CSA) to avoid the carry propagation at each addition operation, 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 drawback, a configurable CSA (CCSA), which could be one full-adder or two serial half-adders, is proposed to reduce the extra clock cycles by half. In the proposed architecture, the column and row-bypassing multipliers examine the number of zeros in either the multiplicand or multiplicator to predict whether the operation requires one cycle or two cycles to complete. As a result, the extra clock cycle can be hidden and high throughput can be obtained.*

Keywords: *modular multiplication, CCSA, skip and detect, bypass multiplier.*

I. Introduction

A. Montgomery Modular multiplication

The Montgomery algorithm for modular multiplication is considered to be the fastest algorithm to compute xy mod n in computers when the values of x, y, and n are large. In this lecture note, we shall describe the Montgomery algorithm for modular multiplication. Suppose we want to compute xy mod n in a computer. We first choose a positive integer, r, greater than n and relative prime to n. The value of r is usually 2^m for some positive integer m. This is because multiplication, division and modulo by r can be done by shifting or logical operations in computers.

Modular multiplication is a fundamental operation in many popular Public Key Cryptography (PKC) algorithms such as RSA [1] and ECC [2, 3]. As the division operation in modular reduction is time-consuming, Montgomery [4] proposed a new algorithm where division is avoided. An integer Z is represented as $Z \cdot R \bmod M$, where M is the modulo and $R = 2^r$ is a radix that is co prime to M. This representation is called Montgomery residue. Multiplication is performed in this residue, and division by M is replaced with division by R. This algorithm can be easily implemented on general purpose processors. However, due to the highly intensive computation, software implementations are often not fast enough.

B.SCS-Based Montgomery Multiplication

To avoid the long carry propagation, the intermediate result
S of shifting modular addition can be kept in the carry-save representation (*SS*, *SC*). However, the format conversion from the carry-save format of the final modular product into its binary format is needed. A 32-bit CPA with multiplexers and registers , 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 will take 32 clock cycles to complete the format conversion of a 1024-bit SCS-based Montgomery multiplication. 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-1multiplexer and one additional register to store the operand *D*. However, they did not present an effective approach to remove the CPA for format conversion and thus this kind of multiplier still suffers from the critical path of CPA.

II. Modular Multiplication Algorithm

A.FCS Based Montgomery Multiplication

The FCS strategy maintains the input and output operands *A*, *B*, and *S* in the carry-save format, denoted as (*AS*, *AC*), (*BS*, *BC*), and (*SS*, *SC*), respectively, to avoid the format conversion, leading to fewer clock cycles for completing a MM.

An energy-efficient FCS-based multiplier is used (denoted as FCS-MMM42 multiplier) in which the superfluous operations of the four-to-two (two-level) CSA architecture are suppressed to reduce the energy dissipation and enhance the throughput. However, the FCS-MMM42 multiplier still suffers from the high area complexity and long critical path delay.

A famous approach to implement modular multiplication in hardware circuits is based on the Montgomery modular multiplication algorithm since it has many advantages. To speed up the encryption/decryption process, many high-speed Montgomery modular multiplication algorithms and hardware architectures employ carry-save addition.

B. *CSA Based Montgomery Multiplication:*

Montgomery multiplication algorithm is the most efficient algorithm available. The main advantage of Montgomery algorithm is that it replaces the division operation with shift operations. During two decades many alternative forms of Montgomery algorithms are introduced. These architectures use carry save addition. The work presented two types of Montgomery algorithms which use Carry Save Adder (CSA). One of the two types used four-to two CSA and the other used five-to-CSA. They had given a brief comparison between these two versions of Montgomery multipliers. They had found that the multiplier using four-to-two CSA architecture has shorter critical path than that of five-to-two CSA multiplier. But extra storage elements and multiplexers are required for 4-to-2 architecture which probably increases the energy consumption. The work in proposed a Montgomery multiplication algorithm using pipelined carry save addition to shorten the critical path delay of five-to-two CSA. This method also required additional pipeline registers and multiplexers which will increase the area.

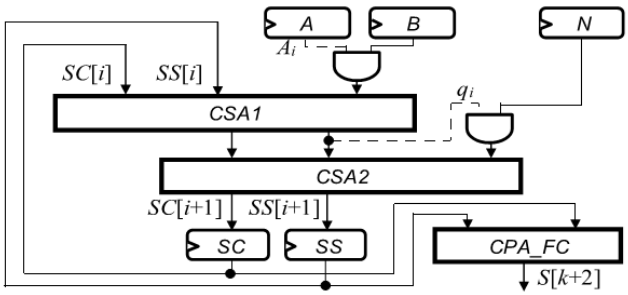

Fig 1.a *CSA Based Montgomery Multiplier*

III. Proposed Montgomery Multiplication

In this section, we propose a new SCS-based Montgomery

MM algorithm to reduce the critical path delay of Montgomery multiplier. In addition, the drawback of more clock cycles for completing one multiplication is also improved while maintaining the advantages of short critical path delay and low hardware complexity.

A. Critical Path Delay Reduction

The critical path delay of SCS-based multiplier can be reduced by combining the advantages of FCS-MM-2 and SCS-MM-2. That is, we can pre-compute $D = B + N$ and reuse the one-level CSA architecture to perform $B+N$ and the format conversion.

The *Zero_D* circuit 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 . 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 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. However, in addition to performing the three-input carry-save additions $k + 2$ 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 $SS + SC$.

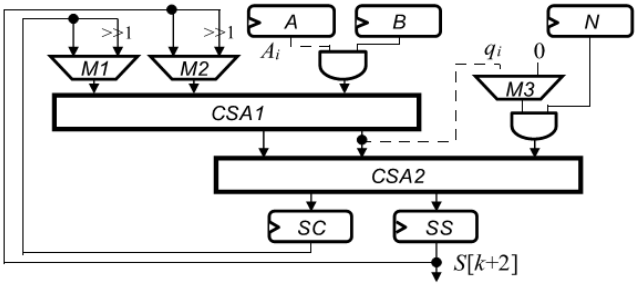

Fig 1.b S*CS Based Montgomery Multiplier*

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 .Each cell is one conventional FA which can perform the three-input carry-save addition. If $\alpha = 1$, CFA is one FA and can perform one three-input carry-save addition (denoted as 1F_CSA). Otherwise, it is two half-adders (HAs) and can perform two serial two-input carry-save additions (denoted as 2H_CSA).Moreover, we modify the 4-to-1 multiplexer *SM3* into a simplified multiplier *SM3* , because one of its inputs is zero, where ~ denotes the INVERT operation. Note that *M3* has been replaced by *SM3* in the proposed one-level CCSA architecture. According to the delay ratio, *T*SM3(i.e., 0.68 × *T*FA*)* is approximate to *T*MUX3 (i.e., 0.63 × *T*FA*)*and *T*MUXI2 (i.e., 0.23 × *T*FA*)* is smaller than *T*XOR2(i.e., 0.34×*T*FA*)*. Therefore, the critical path delay of the proposed one-level CCSA architecture is approximate to that of the one-level CSA architecture.

In addition, we also skip the unnecessary operations for the loop to further decrease the clock cycles for completing one Montgomery MM. The crucial computation for the loop is performing the following three-to-two carry-save addition:

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

where the variable x may be 0, N, B, or D depending on the values of Ai and qi . When $Ai = 0$ and $qi = 0$, x is equal to 0 and $SS[i]0$ must be equal to $SC[i]0$ because the sum of $SS[i]0 + SC[i]0 + x0$ is equal to 0. That is, if $Ai = 0$ and $qi = 0$, then $SS[i]0 = SC[i]0$. Based on this observation, we can conclude that the sum of the carry propagation addition $SS[i+1]k+1:0 + SC[i + 1]k+1:0$ is equal to the sum of the carry propagation addition $SS[i]k+1:1 + SC[i]k+1:1$ when $Ai = qi = 0$ and $SS[i]0 = SC[i]0 = 0$. As a result, the computation of (1) in iteration i can be skipped if we directly right shift the outputs of one-level CSA architecture in the $(i - 1)$th iteration by two bit positions (i.e., divided by 4) instead of one bit position (i.e., divided by 2) when $Ai = qi = 0$ and $SS[i]0 = SC[i]0 = 0$.

Accordingly, the signal skip$i$+1 used in the ith iteration to indicate whether the carry-save addition in the $(i + 1)$th iteration will be skipped can be expressed as

skip$i$+1 =~*(Ai*+1 V *qi*+1 V *SS*[*i* + 1]0*)*…….. (2)

where V 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, $qi+1$ and $Ai+1$ 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

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 ˆ *q* and ˆ*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 skip*i*+1 signal produced in the
i th iteration. That is, *(q, A) = (qi*+2, *Ai*+2*)* if skip*i*+1 = 1.
Otherwise, *(q, A) = (qi*+1, *Ai*+1*)*.

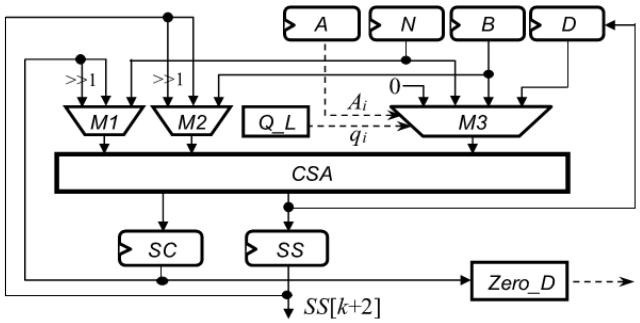

Fig 1.c. Modified SCS based Montgomery multiplication

C. Quotient Pre-computation
As mentioned above, *Ai*+1, *Ai*+2, *qi*+1, and *qi*+2 must
be known in the *i* th iteration for skipping the unnecessary
operation in the (*i*+1)th iteration. It is easy to obtain *Ai*+1 and
Ai+2 in the *i* th iteration. The quotient *qi*+1 can be computed
in the *i* th iteration as follows:

qi+1 = *(SS*[*i* + 1]0 + *SC*[*i* + 1]0 + *Ai*+1 × *B*0*)* mod 2…….. (3)

However, *SS*[*i* +1]0 and *SC*[*i* +1]0 are unavailable until (1) is
completed. Therefore, the critical path of Montgomery multiplier will be largely lengthened if (3) is directly used to produce *qi*+1 in the *i* th iteration.
To avoid this situation, *N*, *B*, and *D* are modified as follows
so that *SS*[*i* + 1]0, *SC*[*i* + 1]0, *qi*+1, and *qi*+2 can be quickly
generated in the *i* th iteration. Since modulus *N* is an odd number and is added in the *i* th iteration only when *qi* is equal to one, it is found that at least a propagated carry 1 is generated since *N*0 is equal to one. Therefore, we can directly employ the value ˆ*N* as shown in (4) instead of *N* to accomplish the process of Montgomery MM. Afterward, ˆ*N*1:0 must be equal to zero.
Moreover, we employ ˆ*B* = 8*B* instead of *B* to ensure that
ˆ*B* 2:0 is equal to zero so that *Ai*+1×*B*0 in (3) can be eliminated and the computation of *qi*+2 can also be simplified. Note that three extra clock cycles at the end of MM for computing division by two are necessary to maintain the correctness of Montgomery MM because *B* is replaced with 8*B*. If *N* and *B* are replaced by ˆ*N* and ˆ*B* , the produced
ˆ*D*1:0 (ˆ*D* = ˆ*N* + ˆ*B)* must be equal to zero.
After *N*, *B*, and *D* are replaced by ˆ*N* , ˆ*B*, and ˆ*D*, we can
ensure that the two LSBs of variable *x* (i.e., *x*1:0*)* in (1) must
be equal to zero. As a result, the carry value *SC*[*i*+1]0 is equal to *SS*[*i*]0 ^*SC*[*i*]0 since *x*0 = 0, where ^denotes the
AND operation. Moreover, the sum value *SS*[*i* + 1]0 is equal to *SS*[*i*]1 ⊕ *SC*[*i*]1 because *x*1 = 0, where ⊕ is the
XOR operation. According to the above results, the logic
expression in (3) for generating *qi*+1 in the *i* th iteration can
be rewritten as
qi+1 = *(SS*[*i*]1 ⊕*SC*[*i*]1*)* ⊕*(SS*[*i*]0 ∧ *SC*[*i*]0*)*…….. (4)

Similar to (3), the quotient *qi*+2 can be generated in the

i th iteration by the following equation:
$qi+2 = (SS[i+2]0 + SC[i+2]0) \bmod 2$……..... (5)

The $qi+2$ will be selected in the *i* th iteration only when
$skipi+1 = 1$. In this case, $Ai+1 = qi+1 = 0$ and $SS[i+1]0 =$
$SC[i+1]0 = 0$ so that $SS[i+2]0 + SC[i+2]0$ is equal to $SS[i+1]1 + SC[i+1]1$. Because $x1 = 0$, $SC[i+1]1 = SS[i]1 \wedge SC[i]1$. Moreover, $SS[i+1]1$ is equal to $SS[i]2 \oplus SC[i]2 \oplus x2$ and $x2$ is equal to 0, $\hat{}N2$, 0, or $\hat{}N2$ when $(Ai, qi) = (0, 0)$, (0, 1), (1, 0), or (1, 1). Therefore, we can obtain that $x2 = qi \wedge \hat{}N2$. As a result, (6) can be simplified and expressed as

$qi+2 = (SS[i]2 \oplus SC[i]2) \oplus (qi \wedge \hat{}N2) \oplus (SS[i]1 \wedge SC[i]1)$...(6)

In addition to $qi+1$ and $qi+2$ can be simplified into (5) and (7), we can also derive a simpler expression for $skipi+1$ in (2). Let $\delta 1 = SS[i]1 \oplus SC[i]1$ and $\delta 0 = SS[i]0 \wedge SC[i]0$,then

$$\begin{aligned} skipi+1 &= \sim(Ai+1 \vee qi+1 \vee SS[i+1]0) \\ &= \sim(Ai+1 \vee (\delta 1 \oplus \delta 0) \vee \delta 1) \\ &= \sim(Ai+1 \vee \delta 1 \vee \delta 0) \\ &= \sim(Ai+1 \vee (SS[i]1 \oplus SC[i]1) \vee (SS[i]0 \wedge SC[i]0)) \end{aligned}$$..(8)

According to (8), we can quickly obtain $skipi+1$ in the *i* th
iteration by $SS[i]1$, $SC[i]1$, $SS[i]0$, and $SC[i]0$.

D. Proposed Algorithm and Hardware Architecture

On the bases of critical path delay reduction, clock cycle number reduction, and quotient pre-computation mentioned above, a new SCS-based Montgomery MM algorithm
(i.e., SCS-MM-New algorithm) 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 $k + 1$.

The hardware architecture of SCS-MM-New algorithm,
denoted as SCS-MM-New multiplier, 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 $skipi+1$, $\hat{}q$, and $\hat{}A$ in the *i* th iteration. Both *M4* and *M5* in 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* are generated by the control part, which are not depicted for the sake of simplicity.

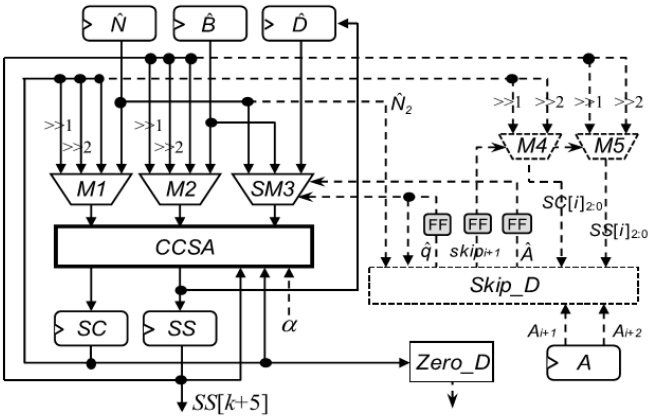

Fig 1.d. *Proposed SCS based Montgomery multiplier*

IV. Experimental Results

A. Analysis of Delay and Area

The maximum delay for generating ˆ *q* through *M4*, *M5*, and *Skip_D* is *T*MUX2 + *T*XOR2 + *T*XOR3 + *T*MUX2. Moreover, *TSM*3 and *T*MUXI2 are less than *T*MUX4 and *T*XOR2. Therefore, the maximum delay of the one-level CCSA architecture for generating *SS* and *SC* is approximate to *T*MUX4 + *T*FA (i.e., 1.71 × *T*FA*)*. As a result, the critical path delay of the SCS-MM-New multiplier is 2.17 × *T*FA, which is less than that of all Montgomery multipliers. Furthermore, the critical path delay of the FCS-MMM42 multiplier in [10] is *T*MUX4 + 2 × *T*FA (i.e., 2.71 × *T*FA*)*, which is much longer than that of the proposed SCS-MM-New multiplier.

On the other hand, the area complexity of SCS-MM-New multiplier is $k \times A$CFA + $5k \times A$REG + $k \times A$SR + $2k \times A$MUX4 + $k \times A$*SM*3. FA is modified into CFA at expense of one 2-input NAND gate and one 2-to-1 MUXI cell. Therefore, *A*CFA is approximate to *A*FA + *A*NAND2 + *A*MUXI2. In addition, *ASM*3 is approximate to *A*NAND2 + *A*MUXI2 + *A*MUX2. As a result, the area complexity of the proposed SCS-MM-New multiplier will be approximate to 9.88*k*×*A*FA. When compared with the previous multipliers, the area of SCS-MM-New multiplier is larger than that of SCS-MM-2 multiplier, but smaller than that of FCS-MM-1 and FCS-MM-2 multipliers. In addition, the area complexity of the FCS-MMM42 multiplier in [10] is $2k \times A$FA + $7k \times A$REG + $2k \times A$SR + $2k \times A$MUX2 + $2k \times A$MUX4 (i.e., 13.28$k \times A$FA*)*, which is much larger than that of the proposed SCS-MM New multiplier. Finally, if the non-configurable one-level CSA architecture is used in the SCS-MM-New multiplier, its area complexity will be reduced to 9.4$k \times A$FA. That is, ∼5% additional area is required to form the CCSA architecture.

B. Implementation Results

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

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.

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.