FPGA-based Design of a Large Moduli Multiplier for Public-Key Cryptographic Systems

Osama Al-Khaleel, Chris Papachristou, Francis Wolff
Case Western Reserve University
Cleveland, Ohio 44106, USA
{oda,cap2,fxw12}@case.edu

Kiamal Pekmestzi
National Technical University
157 80 Athens - Greece
pekmes@microlab.ntua.gr

***Abstract*— High secure cryptographic systems require large bit-length encryption keys which presents a challenge to their efficient hardware implementation especially in embedded devices. Modular multiplication is the core operation in well known cryptosystems like RSA and Elliptic Curve (ECC). Therefore, it is important to employ efficient modular multiplications techniques to improve the overall performance of the cryptographic system. We present a modular multiplier based on the ordinary Montgomery's multiplication algorithm and a new array multiplication scheme to perform the multiplication. The new modular multiplier is scalable and can be used for large bit-lengths. We also implement the modular multiplier into the Virtex4 FPGA devices and we show that our technique has better performance when compared with other schemes. To implement large bit-length multiplications we used a novel partitioning and pipeline folding scheme to fit at least 512-bit modular multiplications on a single FPGA.**

I. Introduction

Modular multiplication, i.e., regular multiplication $\bmod N$, is employed in widely used cryptosystems such as the Rivest-Shamir-Adleman (RSA) [1] and the Elliptic Curve [2] systems. These schemes require large size multiplications in terms of bit-length, say larger than 128 bits, to ensure secure operation. Hardware implementation of these schemes improves performance at the expense of area cost due to the multiplication size.

Among the many proposed modular multiplication algorithms, Montgomery's modular multiplication [3] is the most efficient since it replaces the trial division by the modulus with a series of additions and division by a power of two [4]. Unlike other approaches such as [5], which is based on lookup tables, the computation in Montgomery's algorithm proceeds from the least significant to the most significant digit. This advantage makes Montgomery's multiplication algorithm very efficient for hardware implementation [6]. However, the demand on high security requires very large bit-length operands making efficient hardware design a big challenge.

In many of the modular multiplication algorithms such as those in [4], [7] and [8], the regular multiplication operation is the most demanding among all the operations. Thus, the design of an efficient regular multiplier will have the greatest impact on the overall performance of the modular multiplier especially when operand size is very large ($\geq$ 128 bits in Elliptic Curve Cryptography, up to 2048 bits in RSA).
A large bit-length multiplier to be used in cryptography should meet the following criteria:

- Adaptability: The multiplier should be bit-length scalable and not fixed. Scalability can be achieved by reconfigurability on the fly, which is the major advantage of FPGAs.
- Performance: The multiplier should be very fast to improve the overall performance of the system. This can be achieved by using a fast multiplier architecture.
- Small size: The multiplier should have a reasonable size that can fit in one FPGA and permit for other components in the system to fit in as well. If the size is very large then more than one FPGA will be needed for the entire system. This means more cost and less performance because of signals going out of chip.
  There should be a way to control the size of the multiplier as the bit-length increases. One way to achieve this is by effective partitioning.

In this work, our goal is to build a scalable modular multiplier that can be used in cryptography applications. We employ a modular multiplication algorithm that has been proposed in literature and we propose a modular multiplication architecture that we use to implement this algorithm. However, regular multiplications of large bit-length operands are essential in the algorithm. Thus, we have developed a novel large scale radix-4 multiplexer-based array multiplier, and we use it to perform the multiplication operation in the architecture. We tested the performance of the proposed multiplication scheme by mapping different bit length operands into the FPGA. We used the same FPGA devices to map our scheme and the multiplexer-based array multiplier for k-bit lengths of 16, 32, 64 and 128. Our multiplier shows better performance over a multiplexer-based array multiplier proposed in the literature with minor size penalty. We implemented the modular multiplier shown in Figure 1 in Xilinx FPGA devices, and we compared the results of 128-bit and 256-bit modular multiplier with those referenced in [4], which is also build based on the ordinary Montgomery multiplier. Moreover, the new modular multiplier was implemented in Virtex4 FPGA device for k-bit lengths of 128, 256 and 512. To fit the modular multiplier of large bit-

length in a single FPGA device, we used a partitioning and pipeline folding scheme that allowed us to fit at least 512-bit modular multiplications in a single FPGA.

The rest of this paper is organized as follows:
Section II talks briefly about the encryption and decryption in RSA. Section III discusses our approach to a modular multiplication architecture using a well known modular multiplication algorithm. Section IV presents our novel regular multiplication scheme to build the radix-4 multiplexer-based array multiplier which is used to perform the multiplication operation in the modular multiplication architecture, in Section III. Section V presents the partitioning and pipeline folding technique. We provide experimental results in Section VI and we conclude in Section VII.

II. Encryption/Decryption in RSA

In RSA, the public encryption key consists of two positive integers (E, N). A plaintext message P is partitioned into a sequence of blocks, each of which is M, i.e. an integer between 0 and $N-1$ and is generated by using a padding scheme. M is then raised to the E^{th} power modulo N to obtain the encrypted message C [1], [9] and [10]. On the other hand, to retrieve the original massage M using the private decryption key which is another two positive integers (D, N), the encrypted message C is raised to the D^{th} power modulo N. Both the encryption and decryption equations are given by:

$$C = M^E \mod N \tag{1}$$

$$M = C^D \mod N \tag{2}$$

It is clear from the encryption and decryption equations 1 and 2 that the modular exponentiation is the fundamental operation for encryption. This operation can be realized by repeated modular multiplications using the square and multiply algorithm [11] and [12]. The modular exponentiation is given in Algorithm 1.

Algorithm 1: Modular Exponentiation

Input : M, E, N
Output: $C = M^E (\mod N)$

$C \leftarrow M$;
for $j \leftarrow (k-2)$ **to** $zero$ **do**
  $C \leftarrow C \times C (\mod N)$;
  if $e_j = 1$ **then**
    $C \leftarrow C \times M (\mod N)$;
return C;

III. Modular multiplication architecture

In this section we present a modular multiplication architecture which is based on the ordinary Montgomery's modular multiplication algorithm [3] and [6]. We employ a radix-4 multiplexer-based array multiplier, which is described in Section IV, to perform the regular multiplication operation. Since the regular multiplication operation, as we will see in Section III-A, has main contribution to the overall modular multiplication algorithm, the performance of the regular multiplier dominates the performance of the overall modular multiplier architecture. Thus, we employ a radix-4 multiplexer-based array multiplier which shows better performance as compared to other multiplication schemes, to implement the multiplication operation in the modular multiplication algorithm.

A. Montgomery architecture

Algorithm 2: Montgomery Multiplier

Input : $X, Y, N, N^{'}, r$
Output: $R = X \times Y \times r^{-1} (\mod N)$

$P \leftarrow X \times Y$;
$t \leftarrow P \times N^{'} (\mod r)$;
$t \leftarrow t \times N$;
$R \leftarrow (P + t) \div r$;
if $R \geq N$ **then**
  return $R - N$;
else
  return R;

The pseudocode in Algorithm 2 presents the ordinary Montgomery multiplication algorithm [7] and [4]. In this algorithm, the k-bit integers X, Y and N are the multiplier, the multiplicand and the modulus, respectively. X and Y are N-residue with respect to r where $r = 2^k$. The output of the algorithm is R which is the Montgomery product of X and Y. R is given by :

$$R = X \times Y \times r^{-1} (\mod N) \tag{3}$$

where r^{-1} is the inverse of $r \mod N$, i.e., $r \times r^{-1} = 1 (\mod N)$. The additional quantity $N^{'}$, where $r \times r^{-1} - N \times N^{'} = 1$, is needed to describe the algorithm [7]. Both r^{-1} and $N^{'}$ can be computed using the extended Euclidean algorithm [13] and [7]. As mentioned before, the operand size in cryptosystems is very large to achieve very high security and so a very large multiplier is needed to perform the multiplication in Algorithm 2, which involves three multiplication operations, one addition operation and a conditional subtraction operation. Thus, the performance of the large multiplier has the largest impact on the overall performance. Figure 1 shows the block diagram of the Montgomery modular multiplier architecture that we built based on Algorithm 2. The large multiplier in the architecture, which is implemented using radix-4 multiplexer-based array multiplication scheme, gives the architecture two main advantages. The radix-4 multiplexer-based array multiplier shows better performance than other multiplication schemes as we will see in Section IV and VI and so the performance of the overall modular multiplier is

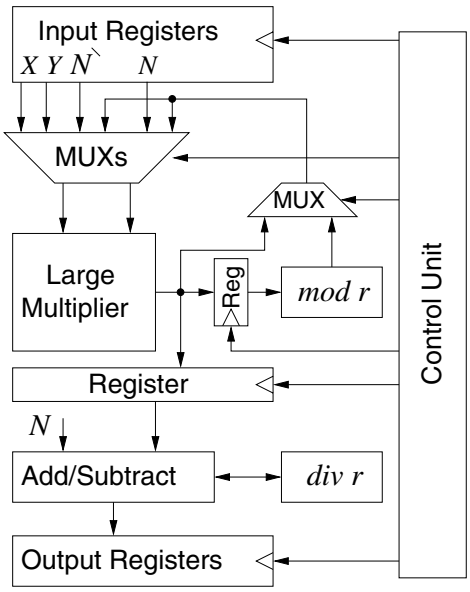


Fig. 1. Montgomery architecture

improved. Moreover, as we will see in Section V, the structure of the radix-4 multiplexer-based array multiplier is amenable to implementing the multiplier in a folded pipeline within one FPGA device as the size of the operands increase to the limit. This occurs when the multiplier does not fit in the FPGA device if implemented with the normal way without using the folded pipeline technique.

IV. Radix-4 multiplication scheme

In this section, we, briefly, introduce our multiplication scheme that we use to develop the multiplier. We present the derivation of the equations and a parallel multiplier architecture that employs this scheme in [14].

A. Multiplication scheme

Assume two k-bit (k is even) numbers X and Y such that

$$X = x_{k-1}x_{k-2}x_{k-3}x_{k-4}.....x_2x_1x_0 \quad (4)$$

$$Y = y_{k-1}y_{k-2}y_{k-3}y_{k-4}.....y_2y_1y_0 \quad (5)$$

Assume $X_{k-3,k-4}$ and $Y_{k-3,k-4}$ are new numbers that are generated by truncating the most two significant bits in X and Y respectively. Then

$$X_{k-3,k-4} = x_{k-3}x_{k-4}.....x_2x_1x_0 \quad (6)$$

$$Y_{k-3,k-4} = y_{k-3}y_{k-4}.....y_2y_1y_0 \quad (7)$$

X and Y can be rewritten as follow

$$X = 2^{k-1}x_{k-1} + 2^{k-2}x_{k-2} + X_{k-3,k-4} \quad (8)$$

$$Y = 2^{k-1}y_{k-1} + 2^{k-2}y_{k-2} + Y_{k-3,k-4} \quad (9)$$

If we define the product of X and Y as P then we can write

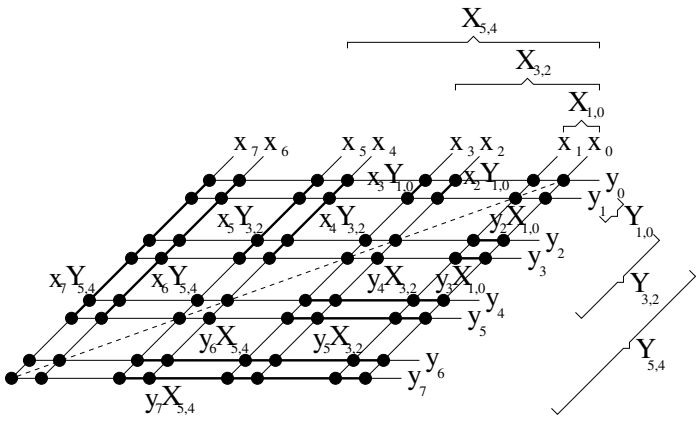


Fig. 2. Schematic illustration of the multiplication algorithm

$$P = XY = \{2^{k-1}x_{k-1} + 2^{k-2}x_{k-2} + X_{k-3,k-4}\} \times \{2^{k-1}y_{k-1} + 2^{k-2}y_{k-2} + Y_{k-3,k-4}\} \quad (10)$$

The expansion of equation 10, taking into consideration that we are processing two bits from each operand at a time as in equations 8 and 9, will lead to the general equation 11 for calculating the product $P = XY$.

$$P = XY = A + B + C \quad (11)$$

Where

$$A = \sum_{i=0}^{(k-2)/2} \{2^{2(2i+1)}x_{2i+1}y_{2i+1} + 2^{2(2i)}x_{2i}y_{2i}\} \quad (12)$$

$$B = \sum_{i=0}^{(k-2)/2} \{2^{2(2i)+1}\{x_{2i}y_{2i+1} + x_{2i+1}y_{2i}\}\} \quad (13)$$

$$C = \sum_{i=1}^{(k-2)/2} \{c1 + c2\} \quad (14)$$

$$c1 = 2^{2i}\{x_{2i}Y_{2i-1,2i-2} + y_{2i}X_{2i-1,2i-2}\} \quad (15)$$

$$c2 = 2^{2i+1}\{x_{2i+1}Y_{2i-1,2i-2} + y_{2i+1}X_{2i-1,2i-2}\} \quad (16)$$

Equation 11 describes a multiplication algorithm and is schematically illustrated in Figure 2. The solid lines connect the partial product bits to distinguish the partial product bits group. If we fold the array in Figure 2 along the diagonal, the multiplication algorithm given by equation 11 can be derived.

We derived general equations to calculate the number of each type of the basic cells for k-bit multiplier. We also derived equations to calculate the number of gates and the number of transistors to be used in building k-bit multiplier. The gate and transistor equations in Table II are derived based on the number of each type of the cells, cell complexity, gates count and transistors count from Table I taken from [15] and [16]

for fair comparison purposes. And then we compared the cell count, the cell complexity, the gates count and the transistors count of the multiplier with those from various multipliers. Our multiplier scheme uses $8.5k^2 + 219k$ gates which is less than the number of gates in the iterative array, the 5-COUNTER cell multiplier and the modified Booth's algorithm in columns two, three and four respectively. The multiplexer-based array multiplier has almost similar gate count dominated by $8.5k^2$. This is also true for the transistor count. However, the experimental results section will show that our scheme is faster than the multiplexer-based array scheme, which is faster than the other three schemes.

Circuit	Gates	Transistors
Full-Adder	12	26
4×1 Mux	5	16
Half-Adder	5	10
XOR Gate	4	6
2-bit CLA	23	50
2×1 Mux	3	6

TABLE I

NUMBER OF GATES AND TRANSISTORS FOR VARIOUS TYPE OF CIRCUITS

V. IMPLEMENTATION USING PARTITIONING AND FOLDED PIPELINE

Partitioning is needed whenever the size of the circuit of the multiplier is large to fit in the FPGA device. The partitioning process is done in a way to have a main partition and other secondary partitions. The main partition is reused to implement any secondary partition by reconfiguring the main partition through control signals. The number of partitions depends on the size of the FPGA device being used and the size of the circuit. The size of the main partition should fit within the FPGA resources. The number of partitions are adjusted until the circuit fits in the FPGA device.

Reusing the main partition can be done by buffering and feeding back the intermediate outputs to the inputs of the main partition, as shown in Figure 3. Multiplexing circuit should be used to choose between the startup inputs and the intermediate outputs. Also, multiplexing is used, during the reuse of the main partition to implement a secondary partition, to isolate any unused component. The multiplexing circuits are controlled by configuration signals that are generated by a control unit. The intermediate feedback outputs are registered by clocked registers.

In this work, the targeted FPGA was Vertex4 FPGA device. We could map the multiplier for up to 128-bit without any need to partition the circuit. However, we had to partition the circuit for higher bit-length. In order to partition the circuit, we first group the basic four cells into two different groups (group A and group B) as shown in Figure 4(a) and 4(b).

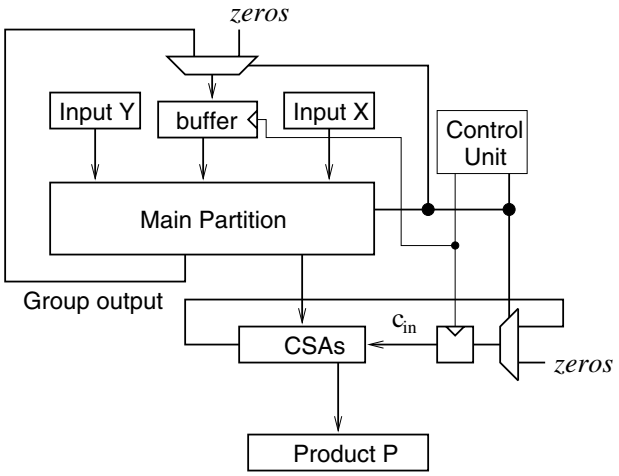
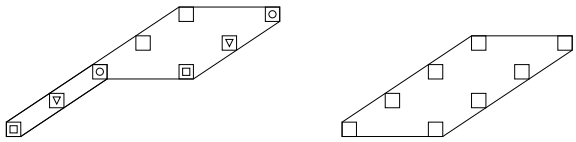


Fig. 3. The overall folded pipeline circuit



(a) Group A (b) Group B

Fig. 4. The grouping

We begin with a 32-bit unpartitioned example shown in Figure 5 where the groups, A and B, are represented by symbols. Group A is represented by an empty circle and group B is represented by a full circle. The CSAs and the connections between the groups are not shown for clarity. To demonstrate the partitioning process in this example, the multiplier was partitioned into four partitions. It turned out that the multiplier can easily be partitioned into different partitions, where the secondary partitions are subsets of the main partition. By having the secondary (i.e. second, third and fourth in Figure 6) partitions being subsets of the main partition, the multiplier has the advantage that the main partition can be reused to implement any of the secondary partitions. The control unit sends configuration signals to control the multiplexing circuits. The four partitions, for this 32-bit example, are shown in Figure 6.

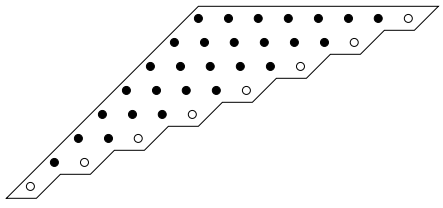


Fig. 5. 32-bit example using cell groups (Group A: empty circle, Group B : full circle)

Multiplier Type	Iterative Array	5-COUNTER Cell	Modified Booth's Algorithm	Multiplexer-based Array	Ours
Number Of Basic Cells	k^2	$k(k+1)/2$	CELL A: $(k+1)^2/2$ CELL B: $(k+1)/2$	CELL I: $k(k-1)/2$ CELL II: k CLA-2: k	Cell1: $k(k-2)/4$ Cell2: $k/2$ Cell3: $k/2$ Cell4: $k/2$ CSA-4: $4k/2$
CELL Complexity	1 Gated FA	2 Gated FA	CELL A: 2x1 MUX, 5 Gates, 1 FA CELL B: 6 Gates, 2 FA	CELL I: 4x1 Mux, 1 FA CELL II: 2 Gates, 2 FA CLA-2: 23 Gates	Cell1: 2(4x1 Mux), 2 FA Cell2 : CSA-2, 2 FA, 3 Gates Cell3: 4x1 Mux, 2 FA, 2 Gates Cell4: 1 FA, 1 Gate CSA-4: 46 Gates
Gate Number	$13k^2$	$13(k^2+3k-2)$	$10k^2+23k+13$	$8.5k^2+40.5k-34$	$8.5k^2+219k$
Transistor Number	$30k^2$	$30(k^2+3k-2)$	$21k^2+52k+31$	$21k^2+89k-73$	$21k^2+487k$

TABLE II
CIRCUIT COMPARISON OF THE VARIOUS MULTIPLIERS

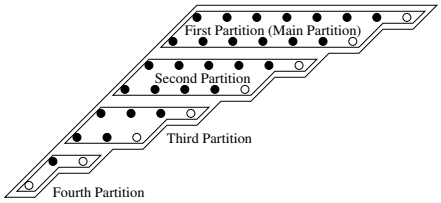


Fig. 6. The four partitions of the 32-bit example

k	Slices	LUTs	Critical Path Delay	Slices Ratio%	LUTs Ratio%
128	20827	31489	195.387ns	97.5 %	98.4%
256	34345	65178	400.184ns	97.0 %	98.4%
512	73629	139611	987.000ns	97.2 %	98.5 %

TABLE III
RESULTS FOR 128-BIT, 256-BIT AND 512-BIT MODULAR MULTIPLIERS

The main partition as well as the secondary partitions are implemented in a folded pipeline scheme within the FPGA device. That is, each partition is instantiated by reconfiguration of the main partition, which is nothing but generating the configuration signals to control the multiplexing circuits. The overall pipeline folding circuit is shown in Figure 3.

VI. EXPERIMENTAL RESULTS

We used the Virtex4 FPGA device to implement the modular multiplier of Figure 1 for different bit-length of 128, 256 and 512. In Table III we listed the number of slices, the number of LUTs and the critical path delay for each case. Table III also shows the ratios percentage of the number of the slices and the LUTs in the regular multiplier to those in the modular multiplier. These ratios show that the regular multiplier has the largest contribution to the overall circuit.

We tested the performance of the proposed multiplication scheme by mapping different bit lengths to the FPGA. We used the same FPGA devises to map our scheme and the multiplexer-based array multiplier for k-bit lengths of 16, 32, 64 and 128. Our multiplier shows better performance over the multiplexer-based array multiplier with minor size penalty. We summarized the results in Table IV. In [16], it is shown that the multiplexer-based array multiplier is faster than other multiplication schemes. This means, the proposed multiplier is faster than those multiplication schemes.

We implemented the modular multiplier in Figure 1 using the Xilinx FPGA devices, and we compared the results of 128-bit and 256-bit modular multiplier with those from [4], which is also build based on the ordinary Montgomery multiplier. In the case of 128-bit, for a fair comparison, we used the same FPGA device that was used by [4]. However, for the 256-bit case we used slower version of the same device. In both cases our modular multiplier showed better performance. These comparisons are given in Table V.

k	Multiplexer-based array multiplier[16]			The proposed array multiplier			
	Slices	LUTs	Critical Path Delay	Slices	LUTs	Critical Path Delay	Speed Up%
16	329	479	20.552ns	714	1233	19.480ns	5.5%
32	1177	2264	40.254ns	1978	3121	25.709ns	56.58%
64	4769	8310	74.643ns	6043	9345	37.616ns	98.43%
128	18964	33014	155.483ns	20315	30977	61.277ns	153.74%

TABLE IV

COMPARING OUR MULTIPLIER WITH THE MULTIPLEXER-BASED ARRAY MULTIPLIER[16]

k	Modular multiplier in[4]		Modular multiplier in Figure1		Speed UP%
	XC2V FPGA device	Critical Path Delay	XC2V FPGA device	Critical Path Delay	
128	P50-7-ff1517	343.8ns	P50-7-ff1517	272.8ns	26.0%
256	P125-7-ff1696	700.5ns	P100-6-ff1704	636.3ns	10.1%

TABLE V

COMPARING MODULAR MULTIPLIER IN FIGURE 1 WITH [4]

VII. CONCLUSION

Modular multiplier with operands of very large bit-length is needed in cryptosystems. In modular multiplication algorithms, like the ordinary Montgomery's multiplier, multiplication is the main operation. To improve the performance of the overall cryptosystem, the modular multiplier should be scalable, small in area and fast. We proposed a modular multiplier to be used in cryptosystems. Our modular multiplier employs a new multiplication scheme, which is based on radix-4 multiplexer-based arrays, to perform the regular multiplication operation. The new modular multiplier showed better performance over other schemes when mapped to Virtex4 FPGA devices. We mapped the modular multiplier to the Virtex4 FPGA devices for key length of 128, 256 and 512. To fit the large modular multiplier in a single FPGA, we used a novel partitioning and pipeline folding scheme that allowed us to fit at least a 512-bit modular multiplier in a single FPGA. To fit modular multipliers of operands with larger bit-lengths, the number of partitions can be adjusted.

REFERENCES

[1] R. L. Rivest, A. Shamir, and L. Adelman, "A method for obtaining digital signatures and public-key cryptosystems," *Communications of the ACM*, vol. 21, pp. 120–126, 1978.

[2] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of computation*, vol. 48, no. 177, pp. 203–209, 1987.

[3] R. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, pp. 519–521, 1985.

[4] C. McIvor, M. McLoone, and J. V. McCanny, "FPGA montgomery modular multiplication architectures suitable for ECCs over $GF(p)$," *IEEE International Symposium on Circuits and Systems*, vol. 3, pp. 509–512, May 2004.

[5] C. W. Wu and Y. F. Chou, "general modular multiplication by block multiplication and table lookup," *Proceedings of the IEEE Int. Symp. Circuits and Systems (ISCAS)*, pp. 295–298, 1994.

[6] M. O. Sanu, E. E. Swartzlander, and C. M. Chase, "Parallel montgomery multipliers," *IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'04)*, pp. 63–72, 2004.

[7] C. K. Koc, T. Acar, and B. S. Kaliski, "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26–33, June 1996.

[8] C. C. Yang, T. S. Chang, and C. W. Jen, "A new RSA cryptosystem hardware design based on montgomery's algorithm," *IEEE Trans. Circuit and Systems II: Analog and Digital Signal Processing*, vol. 45, pp. 908–913, July 1998.

[9] J. H. Hong and C. W. Wu, "Cellular-array modular multiplier for fast RSA public-key cryptosystem based on modified booth's algorithm," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 11, no. 3, pp. 474–484, June 2003.

[10] K. Puttegowda and P. Athanas, "RSA encryption using extended modular arithmetic on the quicksilver COSM adaptive computing machine," *IEEE Symposium on Field Programmable Custom Computing Machines (FCCM 03)*, pp. 305–307, April 2003.

[11] A. Menezes, P. Orschot, and S. Vanstone, *Handbook of Applied Cryptography*. CRC Press, 1997.

[12] S. B. Ors, L. Batina, B. Preneel, and J. Vandewalle, "Hardware implementation of an elliptic curve processor over $GF(p)$," *The 10th Reconfigurable Architectures Workshop (RAW)*, April 2003.

[13] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*. Mass.: Addison-Wesley, 1981.

[14] O. Al-Khaleel, C. Papachristou, F. Wolff, and K. Pekmestzi, "A large scale adaptable multiplier for cryptographic applications," *First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06)*, pp. 477–484, June 2006.

[15] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Mass.: Addison-Wesley, 1985.

[16] K. Z. Pekmestzi, "Multiplexer-based array multipliers," *IEEE Trans. on Computers*, vol. 48, no. 1, pp. 15–23, January 1999.