FPGA VERSUS ASIC IMPLEMENTATION OF RADIX-4 SCALABLE MONTGOMERY MODULAR MULTIPLIER

مقارنة تحقيق ضارب مونتجومري المتبقي مختلف الدقة رباعي القاعدة باستخدام مصفوفة البوابات المبرمجة حقليا مع الدوائر المتكاملة المحددة التطبيق

ATEF A. IBRAHIM[1] , *HAMED A. ELSIMARY*[1], *AMEN M. NASSAR*[2]

[1] Electronics Research Institute, Cairo, Egypt,

[2] Cairo University, Cairo, Egypt

الملخص العربي:

إن تحقيق لوغاريتمات التشفير باستخدام تقنية الدوائر المتكاملة محددة التطبيق يكن ذات مرونة أقل منه في حالة التحقيق بواسطة البرمجة. وحيث أن بروتوكولات السرية لا تعتمد على لوغاريتم معين ، لذا أصبح من المرغوب فيه توفير مرونة بدرجة عالية جدا. فأمثل الحلول التي تجمع بين المرونة العالية وكذلك السرعة العالية ، هو تحقيق لوغاريتمات التشفير بواسطة أجهزة قابلة للبرمجة مثل مصفوفة البوابات المبرمجة حقليا. فالهدف من هذه الورقة العلمية هو مقارنة – من حيث المساحة والسرعة – تحقيق ضارب مونتجومري المتبقي مختلف الدقة رباعي القاعدة باستخدام مصفوفة البوابات المبرمجة حقليا مع الدوائر المتكاملة المحددة التطبيق وذلك لمعاملات ذات دقات مختلفة.

ABSTRACT

Traditional ASIC implementations have the well known draw-back of reduced flexibility compared to software implementations. Since modern security protocols are increasingly defined to be algorithm independent, a high degree of flexibility with respect to the cryptographic algorithms is desirable. A promising solution which combines high flexibility with the speed and physical security of traditional hardware is the implementation of cryptographic algorithms on reconfigurable devices such as FPGA. In this paper we compare – in terms of area and speed- FPGA implementation of radix-4 scalable Montgomery modular multiplier using encoding technique [15] with ASIC implementation for different word sizes of operands. The experimental data were generated using Mentor Graphics CAD tools.

KEYWORDS: **Montgomery Multiplication, Scalability, FPGA Implementation, ASIC Implementation, Cryptography**

1. INTRODUCTION

Modular multiplication is a widely used operation in cryptography. Several well know applications, such as the decipherment operation of the RSA algorithm [1], the Diffie-Hellman key exchange algorithm[2], as well as some applications currently under development, such as the Digital

Accepted September 26, 2006

Signature Standard [3] and elliptic curve cryptography [4], all use modular multiplication and modular exponentiation. The second operation is often implemented by a series of multiplications and additions [6,7,8].

Given the increasing demands on secure communications, cryptographic algorithms will be embedded in almost every application involving exchange of information. Some of these applications, such as smart cards [9] and hand-helds, require hardware restricted in area and power resources [10].

An efficient algorithm to implement modular multiplication is the Montgomery Multiplication algorithm [11], it has many advantages over ordinary modular multiplication algorithms. The main advantage is that the division step in taking the modulus is replaced by shift operations which are easy to implement in hardware [10].

An aspect of cryptographic applications is that very large numbers are used. The precision varies from 128 and 256 bits for elliptic curve cryptography to 1024 and 2048 bits for applications based on exponentiation [12]. Most of the hardware designs for modular multiplication are fixed-precision solutions. That is, the operands can be only of fixed bit-size. Designs that can take operands with an arbitrary precision have been researched in the ASIC [13] and the FPGA [8] realms.

A scalable (variable-precision) Montgomery multiplier design methodology was introduced in [13] in order to obtain hardware implementations. This design methodology allows to use a fixed-area modular multiplication circuit for performing multiplication of unlimited precision operands. The design tradeoffs for best performance in a limited chip area were also analyzed in [13]. Extension of this design methodology to higher radices was introduced in [14].

Traditional ASIC implementations, however, have the well known draw-back of reduced flexibility compared to software implementations. Since modern security protocols are increasingly defined to be algorithm independent, a high degree of flexibility with respect to the cryptographic algorithms is desirable. A promising solution which combines high flexibility with the speed and physical security of traditional hardware is the implementation of cryptographic algorithms on reconfigurable devices such as FPGA.

In this paper we compare – in terms of area and speed - FPGA implementation of radix-4 scalable Montgomery modular multiplier using encoding technique [15] with ASIC implementation for different word sizes of operands. The experimental data were generated using Mentor Graphics CAD tools.

This contribution is structured as follows. In Section 2 we present the radix-4 Montgomery Modular Multilpication algorithm (R4MM). Section 3 presents the overall organization of the modular multiplier that implements the R4MM. Section 4 shows the experemental results, generated using Mentor Graphics CAD tools. Section 5 concludes the work.

2. R4MM ALGORITHM

The notation used in the presented multiple-word Radix-4 Montgomery Multiplication algorithm (R4MM) is shown below (Fig.1).

Fig. 2 shows the R4MM algorithm, which is an extension of the Multiple-Word High-Radix ($R 2^k$) Montgomery Multiplication algorithm (MWR 2^k MM) presented and proved to be correct in [14].

There are two types of recoding applied in the R4MM. The first one (Recoding1) is Booth recoding [16] applied to the multiplier X. This recoding scheme translates conventional radix-4 digits in the set {0, 1, 2, 3} into the digit set {-2, -1, 0, 1, 2} . The recoded digit Z_j is obtained from the radix-4 multiplier digit $X_j = (x_{2j+1}, x_{2j})$ as:

$$Z_j = \mathrm{Re}\,coding1(X_j, x_{2j-1}) = -2x_{2j+1} + x_{2j} + x_{2j-1}$$

where $j = 0,1,2,\ldots,\frac{N}{2}-1$.

In order to make the two least-significant bits of the partial product S all zeros, a multiple of the modulus M, namely qM_jM, is added to the partial product. This step is required to make sure that there are no significant bits lost in the right shift operation performed in step 10. To compute the digit qM_j we

need to examine the two least-significant bits of the partial product S generated in step 4 of the R4MM algorithm .

* X – Multiplier , Y- Multiplicand , M – Modulus, S – Partial product
* N - operands precision
* X_j - a single radix-4 digit of X at position j;
* qM_j - quotient digit that determines a multiple of the modulus M to be added to the partial product S;
* w - number of bits in a word of either Y, M or S;
* $e = \left\lceil \frac{N+1}{w} \right\rceil$ - number of words in either Y, M or S;
* NS - number of stages;
* C_a, C_b - carry bits;
* $(Y^{(e-1)},\ldots,Y^{(1)},Y^{(0)})$ - operand Y represented as multiple words;
* $S^{(i)}_{k-1\ldots 0}$ - bits $k-1$ to 0 of the i^{th} word of S.

Fig.1. Notation

It is shown in [14] that qM_j, as computed in step 5, satisfies the relation $qM_j * M \equiv -S \pmod 4$, which can be rewritten as: $S_{1..0} + qM_j * M_{1..0} \equiv 0 \pmod 4$ and represents the fact that the last 2 bits of S are zeros before the right shift is done in step 10.

It is easy to show from Booth encoding properties that the multiplier X is represented by digits of Z_j [17]. However, it is still necessary to show that Recoding2 (step 5a of R4MM) generates an equivalent result. In order to do that we need to show that $qM'_j \equiv qM_j \bmod 4$. It is well known that conventional quotient digits qM_j are in the set {0, 1, 2, 3}. Applying a recoding function (*Recoding2*) we convert qM_j to another digit set {-1, 0, 1, 2}. The recoding scheme consists in replacing $qM_j = 3$ by the recoded value $qM_j = -1$. It makes the generation of multiples of M less complex. Based on the fact that $-1 \equiv 3 \bmod 4$ it is possible to conclude that $qM'_j \equiv qM_j \bmod 4 \rightarrow qM'_j * M \equiv qM_j * M \bmod 4$. And therefore the application of *Recoding2* generates a result that is congruent to the correct result, modulo M. In fact steps 5 and 5a of the R4MM algorithm can be executed in a single step. Two steps were shown for clarity only.

Step

1: $S := 0$
   $x_{-1} := 0$
2: FOR $j := 0$ TO $N-1$ STEP 2
3: $Z_j = \mathrm{Re}coding1(x_{j+1\ldots j-1})$
4: $(C_a, S^{(0)}) := S^{(0)} + (Z_j * Y)^{(0)}$
5: $qM_j := S^{(0)}_{1..0} * (4 - M^{(0)^{-1}}_{1..0}) \bmod 4$
5a: $qM'_j = \mathrm{Re}coding2(qM_j)$
6: $(C_b, S^{(0)}) := S^{(0)} + (qM'_j * M)^{(0)}$
7: FOR $i := 1$ TO $e-1$
8: $(C_a, S^{(i)}) := C_a + S^{(i)} + (Z_j * Y)^{(i)}$
9: $(C_b, S^{(i)}) := C_b S^{(i)} + (qM'_j * M)^{(i)}$
10: $S^{(i-1)} := (S^{(i)}_{1..0}, S^{(i-1)}_{W-1..2})$
    END FOR;
11: $C_a = C_a$ *or* C_b
12: $S^{(e-1)} := signext(C_a, S^{(e-1)}_{W-1..2})$
    END FOR;

Fig. 2. Multiple-word R4MM algorithm

3. OVERALL ORGANIZATION

The architecture of the modular multiplier that implements the R4MM consists of 3 main blocks; *Datapath (or Kernel)*, *IO & Memory*, and the *Control block*. The computation shown in the R4MM algorithm takes place in the *kernel*. The complete design is presented and discussed in details in[18].

The *kernel* is organized as a pipeline of Processing Elements (PE), separated by registers. Each PE implements one iteration of the R4MM algorithm (steps 3 to 12).

3.1 Radix-4 Processing Element

The radix-4 PE is organized as shown in Fig. 3. The main functional blocks in the PE are: *booth recoding*, *multiple generation (Mult Gen)*, *multi-precision Carry-save adders (MPCSA)*, qM_j *table*, and *registers (shaded boxes)*. The PE operates on w-bit words and for this reason the *Mult Gen* and *MPCSA* modules are capable of storing and transferring carry bits from one word to the next. Shifting and word alignment is done by proper combination of signals and registers at the output of the last MPCSA. The design uses a re-timing technique explained in [14]. More details about these modules and their operation can be found in [18].

The Processing Element (PE) is divided in two sections. The first section (before the register) computes only the two least-significant (LS) bits of each word of $S + Z_j Y$. One can observe that qM_j depends on two LS bits of the data coming from the preceding PE in the pipeline: $(S_{1,0}^{(0)})$ and $Y^{(0)}$, and the recoded digit Z_j . The word size for S needs to be at least 4 bits in order to have the two LS bits of S generated as early as possible for the next PE.

A stage consists of a PE and a register. At each clock cycle, one word of Y , M, SS, and SC is applied as inputs to a stage.

The multiplier digits X_i are transferred to PEs at specific times. The newly computed words of SS and SC, together with words of Y and M, are propagated by each stage to the next stage. This way, small PEs work concurrently to perform several iterations of the R4MM algorithm.

4. EXPEREMENTAL RESULTS

The experimental data were generated using Mentor Graphics CAD tools. The radix-4 design presented in this paper was described in VHDL and simulated in ModelSim for functional correctness.

4.1 ASIC Implementation

Radix-4 design was synthesized using Leonardo synthesis tool for *AMI*05-*fast auto* (0.5 μm *CMOS* technology with hierarchy preserved) provided in the ASIC Design Kit (ADK) from the same company. It has to be noted that the ADK has been developed for educational purposes and therefore cannot be fully compared to technologies used for commercial ASICs, however, it provides a consistent environment for comparison between the designs, and a reasonable approximation of the system performance when using commercial ASIC technology.

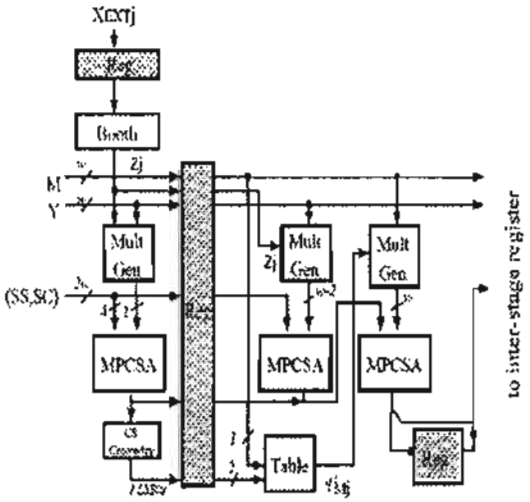

Fig. 3. PE Organization

4.1.1 Area Estimation for Radix-4 Kernel

The area of the kernel depends on the two design parameters: number of stages in the pipeline (NS), and the word size (w) of the operands (Y , M) and the result (S). the total area of the kernel is given by (as mentioned in [18]):

$$A\ker nel_{R4} = 62.86 * NS * w + 146 * NS - 4.875 * w - 13 \quad (1)$$

Table 1 is constructed using Eq. 1. The area estimates are given in terms of 2-input NOR gate.

4.1.2 Time Estimation for Radix-4 Kernel

The total computational time for the kernel is a product of the number of clock cycles it takes and

the clock period. Table 2 shows the critical path delay as a function of the number of stages in the pipeline (NS), as well as the word size (w) of the operands.

Table 1
AREA IN NUMBER OF NOR GATES FOR RADIX-4 KERNEL

NS	Word Size (w)				
	8	16	32	64	128
1	598	1060	1989	3844	7555
2	1246	2212	4146	8013	15747
3	1895	3364	6304	12182	23939
4	2544	4516	8461	16351	32131
5	3193	5667	10617	20520	40323
6	3842	6819	12776	24689	48516
7	4491	7971	14934	28858	56708
8	5139	9123	17091	33027	64900
9	5788	10274	19248	37196	73092
10	6437	11426	21406	41365	81284
11	7086	12578	23563	45534	
12	7735	13730	25721	49704	
13	8384	14881	27879	53873	
14	9033	16033	30036	58042	
15	9681	17185	32194	62211	
16	10331	18337	34351	66380	
20	12926	22944	42981	83056	
25	16170	28703	53769		
30	19415	34461	64557		
35	22659	40220	75344		

A word of Y, M, and S propagates through the pipeline for $(2*NS+1)$ clock cycles. The speed of scanning the bits of X for radix-4 is two bits per stage. Based on these observations, Eq. 2 represents the total number of clock cycles needed for R4MM (as mentioned in [18]).

Table 2
CRITICAL PATH DELLAY FOR RADIX-4 KERNEL (ASIC)

NS	Word Size (w)				
	8	16	32	64	128
1	5..52	5.81	6.25	7.3	6.79
2	5.62	6.13	6.28	7.34	6.84
3	5.62	6.13	6.28	7.34	6.84
4	5.62	6.13	6.28	7.34	6.84
5	5.62	6.13	6.28	7.34	6.84
6	5.62	6.34	6.28	7.34	6.84
7	5.62	6.34	6.28	7.34	6.84
8	5.62	6.34	6.28	7.34	6.84
9	5.62	6.34	6.28	7.34	6.84
10	5.62	6.34	6.28	7.34	6.84
11	5.62	6.34	6.28		
12	5.62	6.34	6.28		
13	6.21	6.34	6.28		
14	6.21	6.34	6.28		
15	6.21	6.34	6.28		
16	6.21	6.34	6.28		
20	6.21	6.34	6.28		
25	6.21	6.34	6.28		
30	6.21	6.34			
35	6.21	6.34			

$$Tclks = \begin{cases} \left\lceil \frac{N}{2*NS} \right\rceil * (2*NS+1) + \left\lceil \frac{N}{w} \right\rceil + 1 & , if \left\lceil \frac{N}{w} \right\rceil \le 2*NS \\ \left\lceil \frac{N}{2*NS} \right\rceil * (\left\lceil \frac{N}{w} \right\rceil + 1) + 2*NS & , if \left\lceil \frac{N}{w} \right\rceil > 2*NS \end{cases} \quad (2)$$

The total computational time is obtained by multiplying $Tclks$ by the corresponding critical path delay (clock period) shown in Table 2, which was obtained from synthesis tools.

4.2 FPGA Implementation

Radix-4 design was synthesized using Leonardo synthesis tool for Xilinx Virtex –II technology.

4.2.1 Area Results

The area in FPGA is given in terms of Configurable Logic Blocks (CLBs) instead of gates as in ASIC implementation. Table 3 shows the area as a function of the number of stages in the pipeline (*NS*), as well as the word size (*w*) of the operands.

Table 3
AREA IN NUMBER OF CLBS FOR RADIX-4 KERNEL

NS	Word Size (*w*)				
	8	16	32	64	128
1	5	11	18	41	54
2	10	19	39	81	134
3	15	29	58	119	214
4	19	40	77	149	307
5	23	49	96	184	
6	28	58	109	220	
7	33	69	128	264	
8	37	78	147	295	
9	42	87	167	320	
10	46	99	185		
11	50	108	203		
12	54	117	222		
13	58	127	242		
14	61	136	261		
15	65	145	280		
16	58	154	300		
20	64	191			
25	84	231			
30	102	275			
35	122	305			

4.2.2 Time Results

Table 4 shows the critical path delay as a function of the number of stages in the pipeline (*NS*), as well as the word size (*w*) of the operands.

Table 4
CRITICAL PATH DELAY FOR RADIX-4 KERNEL (FPGA)

NS	Word Size (*w*)				
	8	16	32	64	128
1	7.42	7.61	8.35	9.35	8.75
2	7.52	7.62	8.38	9.36	8.80
3	7.62	8.13	8.38	9.36	8.80
4	7.65	8.13	8.48	9.36	8.81
5	7.62	8.14	8.48	938	8.83
6	7.63	834	8.48	9.38	8.84
7	7.63	8.35	8.49	9.38	8.85
8	7.64	8.35	8.49	9.39	8.85
9	7.64	8.36	8.51	9.39	8.87
10	7.10	8.44	8.52	9.39	8.87
11	7.20	8.44	8.52		
12	7.21	8.54	8.53		
13	8.20	8.54	8.53		
14	8.22	8.56	8.55		
15	8.22	8.56	8.55		
16	8.31	8.57	8.56		
20	8.32	8.61	8.56		
25	8.32	8.61	8.56		
30	8.33	8.62			
35	8.34	8.63			

5. CONCLUSION

R4MM was implemented on ASIC technology (*AMI05- fast auto)* as well as FPGA (Xilinx Virtex –II) technology. The Montgomery multiplier implemented is a variable-precision solution. FPGA

is selected since it can be easily reconfigured for different word size. Thus, their design area increases correspondingly with the word size used. An FPGA implementation cannot really be compared with an ASIC implementation. However, the timing results suggest that the proposed ASIC implementation can perform as well as the FPGA implementation. Whereas the ASIC implementation cannot be reconfigured, this proposed word size solution design allows the system to work on any precision so long as the precision does not exceed certain limit size.

REFERENCES

[1] L. Adleman and A. Shamir, "A method for obtaining digital signature and public-key cryptosystems," *Comm. ACM*, vol. 21, no. 2, pp. 120-126, February 1978.

[2] M..E. Hellman , "New directions on cryptography," *IEEE transactions on Information Theory*, vol. 22, pp. 644-654, November 1976.

[3] National Institute for Standards and Technology, "Digital signature standard (dss)," Tech. Rep., FIPS PUB 186-2, January 2000.

[4] N. Koblitz, "Elliptic curve cryptosystems.," *Mathematics of Computation*, vol. 48, no. 177, pp. 203-209, January 1987.

[5] A.J. Menezes, *Applications on finite fields.*, Kluwer Academic Publishers, Boston, MA, 1993.

[6] B.S. Kaliski, Ç.K. Koç, and T. Acar, "Analysing and comparing Montgomery multiplication algorithms," *IEEE Micro*, vol. 16, no. 3, pp. 26-33, June 1996.

[7] T. Hamano, "O(n)-depth circuit algorithm for modular exponentiation," in *IEEE 12th Symposium on Computer Arithmetic*. 1995, pp. 188-192, IEEE Computer Society Press, Los Alamitos, CA.

[8] C. Paar and T. Blum, "Montgomery modular exponentiation on reconfigurable hardware," in IEEE 14th Symposium on Computer Arithmetic. 1999, pp. 70-77, IEEE Computer Society Press, Los Alamitos, CA.

[9] D. M. Raihi and D. Naccache, "Cryptographic smart cards," IEEE Micro, vol. 16, no. 3, pp. 14-23, June 1996.

[10] G. Todorov, "ASIC design, implementation and analysis of a scalable high-radix Montgomery multiplier," Master thesis, Oregon State University, USA, December 2000.

[11] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of Computation*, vol. 44, no. 170, pp. 519-521, April 1985.

[12] Ç.K. Koç, E. Savas, and A. F. Tenca, "A scalable and unified multiplier architecture for finite fields GF(p) and FG(2^n)," in *Cryptographic Hardware and Embedded Systems*. 2000, Lecture Notes in Computer Science, Springer, Berlin, Germany.

[13] Ç.K. Koç and A. F. Tenca, "A word-based algorithm and architecture for montgomery multiplication," in *Cryptographic Hardware and Embedded Systems*, C. Paar and Ç. Koç , Ed. 1999, number 1717 in Lecture Notes in Computer Science, pp. 94-108, Springer, Berlin, Germany.

[14] A. F. Tenca, G. Todorov, and Ç.K. Koç, "High-radix design of a scalable modular multiplier," in *Cryptographic Hardware and Embedded Systems* - CHES 2001, Ç.K. Koç and C. Paar, Eds. 2001, Lecture Notes in Computer Science, No. 1717, pp. 189-206, Springer, Berlin, Germany.

[15] L. A. Tawalbeh, A. F. Tenca, and Ç.K. Koç, "A radix-4 design of a scalable modular multiplier with recoding techniques,". *IEEE Potentials*, to appear, 2005.

[16] A. D. Booth, "A signed binary multiplication technique," *Q. J. Mech. Appl. Math.*, vol. 4, no. 2, pp. 236-240, 1951.

[17] G. Todorov, "ASIC design, implementation and analysis of a scalable high-radix Montgomery multiplier," Master thesis, Oregon State University, USA, December 2000.

[18] L. A. Tawalbeh, "Radix-4 ASIC Design of a Scalable Montgomery Modular Multiplier using Encoding Techniques," M.S. thesis, Oregon State University, USA, October 2002.