A modified radix-2 Montgomery modular multiplication with new recoding method

Kooroush Manochehri[a], **Babak Sadeghian**[b], **and Saadat Pourmozafari**[c]

Department of Computer Engineering and IT, Amirkabir University of Technology, Tehran, IRAN

a) *kmanochehri@ce.aut.ac.ir*

b) *basadegh@ce.aut.ac.ir*

c) *saadat@ce.aut.ac.ir*

Abstract: Montgomery modular multiplication algorithm is commonly used in implementations of the RSA cryptosystem and other cryptosystems with modular operations. Radix-2 version of this algorithm is simple and fast in hardware implementations. In this paper this algorithm is modified with a new recoding method to make it simpler and faster. We have also implemented this new algorithm with carry save adders. Results show that, in average the proposed algorithm has about 47% increase of data throughput with maximum 7% increase of hardware area comparing with conventional algorithm.

Keywords: Montgomery, modular multiplication, radix-2, carry save adder, recoding

Classification: Science and engineering for electronics

References

[1] P. L. Montgomery, "Modular multiplication without trial division," *Mathematics of computation*, vol. 44, pp. 519–521, 1985.

[2] K. Manochehri, S. Pourmozafari, and B. Sadeghian, "Improved RNS for RSA hardware implementation," *The CSI journal of computer science and engineering*, vol. 2, pp. 31–39, 2004.

[3] M. Huang, K. Gaj, S. Kwon, and T. Elghazawi, "An optimized hardware architecture for the Montgomery multiplication algorithm," *LNCS4939*, pp. 214–228, 2008.

[4] Y. Fan, T. Ikenaga, and S. Goto, "A high speed design of Montgomery multiplier," *IEICE Trans. Fundamentals*, pp. 971–977, 2008.

[5] C. McIvor, M. McLoone, J. V. McCanny, A. Daly, and W. Marnane, "Fast Montgomery modular multiplication and RSA cryptographic processor architectures," *Proc. 37th Annual Asilomar Conference on Signals, Systems and Computers*, pp. 379–384, 2003.

[6] K. Manochehri and S. Pormozafari, "Modified radix-2 Montgomery modular multiplication to make it faster and simpler," *IEEE Computer Society Int. Conf. Inf. Technol.: Coding and Computing*, vol. 1, pp. 598–602, 2005.

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

[7] K. Manochehri, S. Pourmozafari, and B. Sadeghian, "A new operator for multi-addition calculations," *Advances in Computer Science and Engineering, Communications in Computer and Information Science*, Springer, vol. 6, pp. 938–941, 2009.
[8] K. Manochehri, S. Pourmozafari, and B. Sadeghian, "Very fast multi operand addition method by bitwise subtraction," *IEEE Computer Society Proc. Fifth Int. Conf. Inf. Technol.*, pp. 1240–1241, 2008.

1 Introduction

RSA is the most widely used public-key cryptosystem. An RSA operation is a modular exponentiation as $\mathrm{C} = \mathrm{A}^{\mathrm{e}} \pmod{\mathrm{n}}$.

Montgomery [1] and RNS are the most widely used to enhance the computing speed. The Montgomery multiplication algorithm is an efficient method for modular multiplication with an arbitrary modulus, particularly suitable for implementation on general-purpose computers. The radix-2 Montgomery method is based on an ingenious representation of the residue class modulo N, and replaces division by N operation with division by power of 2. However, there have been various attempts to improve its hardware implementations performance [2, 3, 4, 5].

Our previous paper [6] proposed a new radix-2 Montgomery algorithm for RSA cryptosystem. But in this paper, with a new recoding method, a new radix-2 Montgomery multiplication algorithm is proposed that can be used not only for the implementation of RSA cryptosystem, but also for all other cases. In this new algorithm one step of the main loop is removed that leads to a faster algorithm. This new recoding method was first introduced in references [7, 8], here we describe it again and proposed our general recoding method and based on it, a modified Montgomery algorithm is proposed. The results show a major improvement in its performance as would be described in section 6.

2 Radix-2 Montgomery multiplication

The radix-2 version of Montgomery multiplication algorithm that calculates the Montgomery product of A and B is summarized in the pseudo code below [5].

```
Radix-2 Montgomery Multiplication (A, B, n)
$S[0] = 0;$
for $i$ in 0 to $k-1$ loop
    $q_i = (S[i]_0 + A_i.B_0)$ mod 2;
    $S[i+1] = (S[i] + A_i.B + q_i.n)$ div 2;
end loop;
return $S[k]$;
```

In this algorithm $\mathrm{A} = \mathrm{a}.2^{\mathrm{k}} \pmod{\mathrm{n}}$ and $\mathrm{B} = \mathrm{b}.2^{\mathrm{k}} \pmod{\mathrm{n}}$ and k is the number of bits of the operands. $\mathrm{S[i]}_0$ stands for the LSB of S[i] and $\mathrm{A_i}$ stands for the ith bit of A. The output of this algorithm is $\mathrm{A.B.2^{-k}} \pmod{\mathrm{n}}$.

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

The critical delay of this algorithm occurs during the calculation of the S values given by the three inputs addition $S[i+1] = (S[i] + A_i.B + q_i.n)$.

3 Our recoding method

References [7, 8] introduce a new recoding method that can be employed for many arithmetic algorithms, such as multi-addition or multi-subtraction. In this section this new recoding method is described.

A new operator is defined as bitwise subtraction, and is shown with Θ to recode the operand. Through our proposed recoding, an X can be recoded with two numbers x1 and x2 such that $X = x1 \Theta x2$. Thus our recoding method results binary signed digits. Table I shows relationship between bits of X, x1 and x2 for selecting proper coding. These selections are done base on the target algorithms, to enhance their performances.

Table I. Our recoding method

	Available coding	
X=x1Θx2	x1	x2
0	0	0
0	1	1
1	1	0
1	0	-1

We can use this recoding to enhance multi-addition or multi-subtraction calculations. This is done in references [7, 8].

3.1 Generalization of recoding method

In the previous section, a recoding method with an operator named as bitwise subtraction was described. In general, we may have any operator for recoding an operand, however this operator should yield the result and be easy to be accomplished, or at least be easier than the main operator. For instance assume that we want to compute (X op Y) and we have a set of operators as OP_SET:

$$OP\_SET = \{all\ operations\ that\ can\ be\ computed\ easily\}$$

To do op, we can recode X as X = x1 op1 x2, that op1 is one of the operations chosen from OP_SET. To select op1, we should consider that it should be an easily computed operator and it can help us to speedup the calculation X op Y. Finally to obtain the result, (x1 op1 x2) op Y should be calculated.

Our generalized recoding method may be used in many arithmetic algorithms to enhance its performance. So, the suitable selection of op1 is open problem for each algorithm. If op1 has a property with the relation (x1 op1 x2) op Y = (x1 op Y) op1 x2, we might have a great performance enhancement, depending on x1 representation. For instance, our references [7, 8] shows that bitwise subtraction can speedup the multi-addition speed because $(x1\Theta x2) + Y = (x1 + Y)\Theta x2$. Bitwise subtraction or some other operations

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

can act the rule of op1. For instance, CSA is achieved with our generalized recoding method, if op1 is defined as bitwise addition and op as full addition.

4 Modified radix-2 Montgomery multiplier

As mentioned, the critical delay for radix-2 Montgomery multiplier is the delay of step 2 of the loop. One parameter of this delay is to calculate q_i. Then q_i is multiplied with n, and the result is added to the summation. This delay becomes more important when CSA architecture is used.

In order to calculate q_i, the LSB of previous result, $S[i]_0$, is added to $A_i.B_0$. If B_0 is equal to zero this step can be removed and q_i would be equal to the LSB of the previous result. When B is an even number, this condition is satisfied. We can change the operands to our desired forms, through our recoding method. So, we can recode B as $x1 \Theta x2$. Setting $x1 = B$ and $x2 = 0$ when B is even, and setting $x1 = B - 1$ and $x2 = -1$ when B is odd, make the operand even. After this recoding, x1 should be the input of the loop instead of B. With this assumption, $x1_0$ is zero and the first step of the loop can be removed. The result has no changes if B is even but when B is odd the result should be corrected as follows.

The final result of radix-2 Montgomery algorithm is $A.B.2^{-k} (\mathrm{mod}\ n)$. If we put B as $x1 = B - 1$, the result will be $A.(B - 1).2^{-k} (\mathrm{mod}\ n) = A.B.2^{-k} - A.2^{-k} (\mathrm{mod}\ n)$. To correct the result we can add $A.2^{-k} (\mathrm{mod}\ n)$ to the output of algorithm to reach the desired value $A.B.2^{-k} (\mathrm{mod}\ n)$. As we know $A = a.2^k\ (\mathrm{mod}\ n)$, so $A.2^{-k}\ (\mathrm{mod}\ n)$ is equal to $a.2^k.2^{-k}\ (\mathrm{mod}\ n) = a$. After the completion of the loop we can add 'a' to the result to correct it. So we can have the following algorithm:

Modified Radix-2 Montgomery Multiplication (a, A, B, n)

```
$S[0] = 0;$
$if\ B_0 = 1\ then\ x1 = B - 1,\ x2 = -1$
                 $else\ x1 = B,\ x2 = 0;$
$for\ i\ in\ 0\ to\ k - 1\ loop$
    $S[i + 1] = (S[i] + A_i.x1 + S[i]_0.n)\ div\ 2;$
$end\ loop;$
$q = (S[k]_0 \Theta (x2.a_0))\ mod\ 2$
$S[k + 1] = (S[k] \Theta (x2.a) + q.n)\ div\ 2;$
$return\ S[k + 1];$
```

We can change the term $S[k]\Theta(x2.a)$ to bitwise addition of S[k] with 'a', when B is odd, and with zero, when B is even. This can be done through the existing CSA architecture with the new inputs.

As we know, if the final result of Montgomery multiplier be less than 2n, the final reduction is removed [5]. If B is even, in the final iteration, we have $S[k] < 2n$ and then $S[k + 1] < (2n + n)/2 < 2n$. Also if B is odd we have $S[k + 1] < (2n + a + n)/2 < 2n$. Note that 'a' is always less than n.

The proof of the correctness of the result is as follows. As we know from the radix-2 Montgomery multiplication algorithm, by modified algorithm we have $S[k] = A.x1.2^{-k}\ (\mathrm{mod}\ n)$. If B is even, by the modified algorithm,

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

we have $S[k+1] = A.B.2^{-(k+1)} \pmod n$ (Note: $x1 = B$ and $x2 = 0$). If B is odd, we have $S[k+1] = A.(B-1).2^{-(k+1)} + a.2^{-1} \pmod n$ (Note: $x1 = B-1$ and $x2 = -1$), therefore $S[k+1] = A.(B-1).2^{-(k+1)} + A.2^{-k}.2^{-1} = A.(B-1).2^{-(k+1)} + A.2^{-(k+1)} = A.B.2^{-(k+1)} \pmod n$. So, to convert back from n-residue to normal number we should multiply the result with $2^{-(k+1)}$ instead of 2^{-k} in Montgomery multiplication algorithm.

5 Implementation

To implement our modified Radix-2 Montgomery multiplication algorithm, we use CSA architecture. Our modified Radix-2 Montgomery multiplication algorithm can be implemented with employing 5-to-2 CSA logic [5, 6] as in the following CMRMM algorithm:

```
5-to-2 CSA modified radix-2 Montgomery multiplication (a, A1, A2, B1, B2, n)
$S1[0] = 0;$
$S2[0] = 0;$
$if\ B1_0 + B2_0 = 1\ (mod\ 2)\ then$
    $x1\_1 = make\_even(B1), x1\_2 = make\_even(B2), X2 = -1$
$else$
    $x1\_1 = B1, x1\_2 = B2, x2 = 0;$
$for\ i\ in\ 0\ to\ k-1\ loop$
    $Si_0 = (S1[i]_0 + S2[i]_0)\ mod\ 2;$
    $S1[i+1], S2[i+1] =$
        $CSR(S1[i] + S2[i] + A_i.(x1\_1 + x1\_2) + Si_0.n)\ div\ 2;$
$end\ loop;$
$q = (S1[k]_0 + S2[k]_0 \Theta(x2.a_0))\ mod\ 2$
$S[k+1] = (S1[k] + S2[k] \Theta(x2.a) + q.n)\ div\ 2;$
$return\ S1[k+1],\ S2[k+1];$
```

Note that the input operand A and B and the output product S are represented in carry save format as A1 and A2, B1 and B2, and S1 and S2 respectively. CSR stands for carry save representation. Barrel Register Full Adder (BRFA) can be used to compute Ai, where Ai stands for the ith bit of A [5]. In this algorithm, make_even sets the least significant bit to zero to change the operand to even number. A four-to-two CSA module [5] can be used to calculate $S[k+1]$, but the existing five-to-two CSA architecture can also be used, as:

$$S[k+1] = CSR(S1[k] + S2[k] + 0\ \Theta(x2.a) + q.n)\ div\ 2.$$

This implementation needs one clock cycle for resetting S1[0] and S2[0], and also one extra clock cycle for the last calculation after the loop. Thus this algorithm may be executed in $k+2$ clock cycles whereas the standard algorithm has $k+1$ clock cycles but with less frequency.

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

6 Results

In order to compare the two algorithm, they are implemented for ASIC and FPGA technologies with VHDL language and synthesized by means of Leonardo Spectrum 2002 tool. For ASIC synthesis, CMOS 0.6 micron meter library and for FPGA synthesis, Xilinx Virtex2 series are used. The CSA Montgomery architecture from reference [5] is used to implement the conventional radix-2 Montgomery multiplier. So, the operands are in CSA format and we don't have any changes in the number of inputs for our new algorithm implementation except input 'a'. The reported area contains the area used for this extra input. The results are shown in Table II. Note that area for FPGA is measured in term of slices and for ASIC in term of gates. Comparing with conventional Montgomery algorithm, there are only 7.8% additional gates for area in ASIC designs, but throughput rate is increased something between 30%-40%, depending on the bit lengths. For FPGA designs, these results are still improved. Throughput is increased something between 50%-66% and area is only increased something between 0.12%-0.23%.

Table II. Results of synthesis

		Radix-2 Montgomery multiplier		New Radix-2 Montgomery Multiplier	
	Bit Length (k)	Area (Gates/Slices)	Throughput Rate (Mb/s)	Area (Gates/Slices)	Throughput Rate (Mb/s)
ASIC	256	8206	108.77	8847	147.44
	512	16282	108.08	17556	141.44
	768	24532	100.06	26445	139.83
	1024	32490	99.6	35032	139.62
FPGA	256	1574	64.24	1576	96.94
	512	3408	45.51	3416	75.80
	768	5177	45.44	5188	74.90
	1024	6814	44.95	6830	68.86

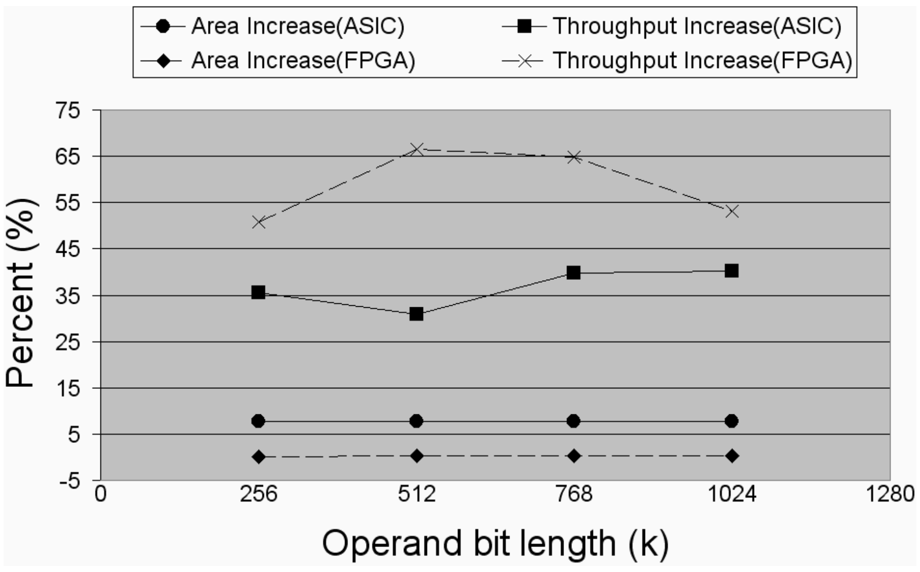

Fig. 1. %Throughput/Area increase using new Montgomery algorithm

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010

Fig. 1 provides a graphical representation of the percentage increases in data throughput rate and area.

This figure shows that, the percentage increase in throughput is higher than percentage increase in area.

7 Conclusion

Recoding method can help in enhancing the computation speed. In this paper, a new recoding method is presented and a generalized recoding method is proposed. Based on this new recoding method, a new algorithm for radix-2 Montgomery multiplier is introduced that achieves greater performance than standard multiplier, when performance is defined as area $\times$ time. RNS or the other implementations that are used Montgomery multiplier can use this new algorithm to enhance their performance.

This new algorithm also can achieve better performance for sequential software implementations, as one of its steps has been removed and so it can be run faster.

© IEICE 2010
DOI: 10.1587/elex.7.513
Received December 25, 2009
Accepted February 26, 2010
Published April 25, 2010