HIGH-SPEED IMPLEMENTATION OF SM2 BASED ON FAST MODULUS INVERSE ALGORITHM

Wei Li[1]*, Juhua Liu*[1]*, Guoqiang Bai*[2*]
1 Institute of Microelectronics, Tsinghua University, Beijing, China
2 National Laboratory for Information Science and Technology, Tsinghua University, Beijing, China
Corresponding author baigq@tsinghua.edu.cn

ABSTRACT

In this paper, we explore the fast modulus inverse algorithm and its implementation. For the first time, we proposed a radix-8 modulus algorithm to speed up the point multiplication in SM2 public key cryptographic algorithm, which is established as the ECC standard of China for commercial applications released by the State Cryptographic Administration of China in December 2010. The critical path delay of our hardware implementation of SM2 is the delay of a one-cycle 256-bit multiplier, which is difficult to get a further reduction. The possibility of further optimization is reducing the number of cycles needed by the binary modulus inverse without changing the critical path delay when converting the Jacob coordinates back to affine coordinates. The radix-8 binary inverse algorithm can reduce the number of cycles significantly by 33.2% on average compared with the radix-4 binary inverse algorithm, which needs 256 cycles at most to complete the conversion.

Keywords—SM2; ECC; radix-8 inverse; high speed; hardware implementation

INTRODUCTION

ECC provides better security compared with other public key cryptographic algorithms, such as RSA, with the same key size [1]. SM2 public key cryptographic algorithm, which is released by the State Cryptography Administration of China in December 2010, is established as the ECC standard of China for commercial applications and is widely used in personal identification, digital signature, finance authentication, etc. SM2 is defined over a Galois field GF(*p*), where the $p = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$, and p is a very special pseudo-Mersenne prime that can be used to compact the process of modulo reduction significantly, more details about SM2 can be found in [2].

Point multiplication, defined as $kP = P + P + ... + P$, where $k \in [0, p-1]$, $P \in GF(p)$ and P is the base point, is the most time-consuming operation in SM2 (more than 90%) [3]. It consists of repeated point addition (PADD) and doubling (PDBL) operations of base point P while performing the point multiplication. However, algorithms to perform PADD and PDBL differ by the choice of coordinate system, leading a different quantity of arithmetic operations, as listed in TABLE I. (A→Affine, P→Projective, J→Jacob, C→Chudnovsky, I→Inversion, M→Multiplication, S→Square)

TABLE I OPERATION COUNTS OF PADD AND PDBL

PADD		***PDBL***	
A+A→A	1*I*,2*M*,1*S*	*2A→A*	1*I*,2*M*,2*S*
P+P→P	12*M*,2*S*	*2P→P*	7*M*,3*S*
J+J→J	12*M*,4*S*	*2J→J*	4*M*,4*S*
C+C→C	11*M*,3*S*	*2C→C*	5*M*,4*S*
J+A→J	8*M*,3*S*		
C+A→C	8*M*,3*S*		

As shown above that *2J→J* for the PDBL and *J+A→J,* mixed Jacobian-affine coordinate, for the PADD is the best choice to get a high-speed architecture. For a further clarification, we give the detailed formulas of PADD and PDBL operation in TABLE II.

TABLE II PADD AND PDBL FORMULAS

$P_3 \leftarrow P_1 + P_2$
$X_3 = (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2)$ $Y_3 = (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3$ $Z_3 = (X_2Z_1^2 - X_1)Z_1$
$P_3 \leftarrow 2P_1$
$X_3 = (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2$ $Y_3 = (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4$ $Z_3 = 2Y_1Z_1$

Once the point multiplication is done, we have to transfer the results to standard affine coordinate system form Jacobian coordinate system, in which an inversion is necessary.

HARDWARE ARCHITECTURE

This section we explore the high-speed hardware implementation of SM2, specifically we will give the hardware architecture of the point multiplication algorithm, as shown in Fig 1. In this architecture we input the base point $P = (X_2, Y_2, Z_2)$ then compute the kP and return the result in affine coordinate (x, y).

Multiplier: A 256-bit multiplier and a two-stage pipeline are introduced to our architecture to get a best trade-off between the number of cycles needed to complete the point multiplication and the minimum critical path of the datapath. In this case, the critical path of this hardware architecture is the delay of the 256-bit one-cycle multiplier [4].

Reduction: A two stage pipeline is applied to the modular-multiplication and a new reduction module is designed to best accelerate the point multiplication. To get a more balanced pipeline, we combined the modular

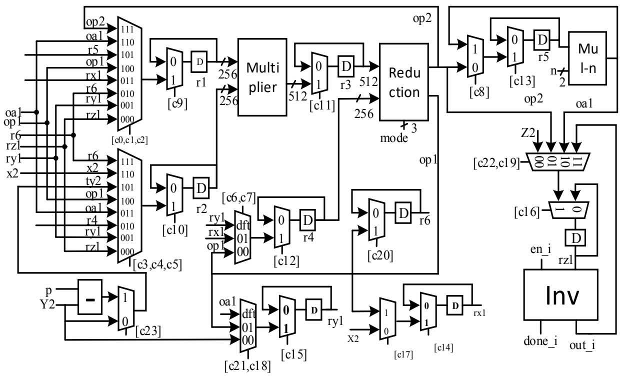

Fig.1 hardware implementation architecture of point multiplication of SM2 algorithm. A 256-bit multiplier and fast modulus algorithm (Inv) are used.

addition and the modular abstraction together with the modular reduction to obtain a new reduction module, which is controlled by signal *mode*, and the specific operation according to the value of *mode* is shown in TABLE III.

Mul-n: This module is designed to compute the 2,3,8 multiple of its input according to the selection signal *n*.

Inv: The module ***Inv*** coverts the result in Jacob coordinates to affine coordinates.

TABLE III OPERATIONS OF THE NEW REDUCTION MODULE

mode	***corresponding operations***
000	$out_1=(in_1\%p-in_2)\%p, out_2=(in_1\%p+in_2)\%p$
001	$out_1=(in_2-in_1\%p)\%p, out_2=(in_1\%p+in_2)\%p$
010	$out_1=(in_1\%p-in_2)\%p, out_2=(1.5in_2\%p-in_1)\%p$
011	$out_1=in_1\%p, out_2=in_1\%p$
100	$out_1=3in_1\%p, out_2=3in_1\%p$
101	$out_1=2in_1\%p, out_2=2in_1\%p$

BINARY MODULUS INVERSE ALGORITHM

As depicted in the Fig 1, the inversion module ***Inv*** that coverts the Jacob coordinates to affine coordinates is another bottleneck of the proposed architecture [5]. So, to get a further optimization on the number of cycles, we proposed a radix-8 binary inverse algorithm. As the best of our knowledge, the inverse of $a, a\in[1,p-1]$,is defined by $b=a^{-1}\bmod p$, $b\in[1,p-1]$, where $a\cdot b=1\bmod p$. Generally, we compute the $a^{-1}\bmod p$ as follow:

$$a\cdot x_1=u\bmod p \qquad (1)$$

$$a\cdot x_2=v\bmod p \qquad (2)$$

We get the inversion a^{-1} whenever $u=1$ ($a^{-1}=x_1$) or $v=1$ ($a^{-1}=x_2$) by right shifting u or v (binary representation), or right shifting after subtracting u from v , *if* $v>u$ ($u-v$, *if* $u>v$) on the condition of keeping the two equations Eq(1) and Eq(2) always being true during the whole inversion process. Conventional binary radix-2 modulus inverse algorithm deals with one bit once a clock

TABLE IV
FAST RADIX-4 MODULUS INVERSE ALGORITHM

Radix-4 Modulus Inverse Algorithm on *GF(p)*

```
Input: prime $p, a\in[1,p-1]$
Output: $a^{-1}\bmod p$
1.$u\leftarrow a, v\leftarrow p, x_1\leftarrow 1, x_2\leftarrow 0$
2. if $v>0$, repeatedly execute
  2.1 $c=u[1:0], d=v[1:0]$
  2.2 if $c=2'b00$, execute
      $u\leftarrow u/4,\ x_1\leftarrow x_1/4$
  2.3 else if $d=2'b00$, execute
      $v\leftarrow v/4,\ x_2\leftarrow x_2/4$
  2.4 else if $c=d$, execute
      2.4.1 if $u>v, u\leftarrow(u-v)/4,\ x_1\leftarrow(x_1-x_2)/4$
      2.4.2 else, $v\leftarrow(v-u)/4,\ x_2\leftarrow(x_2-x_1)/4$
  2.5 else if $c[1:0]=2'b10$, execute
      2.5.1 if $u/2>v, u\leftarrow(u/2-v)/2,\ x_1\leftarrow(x_1/2-x_2)/2$
      2.5.2 else, $v\leftarrow(v-u/2)/2,\ x_2\leftarrow(x_2-x_1/2)/2$
  2.6 else if $d[1:0]=2'b10$, execute
      2.6.1 if $u>v/2, u\leftarrow(u-v/2)/2,\ x_1\leftarrow(x_1-x_2/2)/2$
      2.6.2 else, $v\leftarrow(v/2-u)/2,\ x_2\leftarrow(x_2/2-x_1)/2$
  2.7 else if $u\geq v$, execute
      $u\leftarrow(u-v)/2,\ x_1\leftarrow(x_1-x_2)/2$
      $v\leftarrow(v-u)/2,\ x_2\leftarrow(x_2-x_1)/2$
3.return $x_1$
```

cycle and radix-4 modulus inverse algorithm, as shown in TABLE IV, deals with two bits once a clock cycle (the LSBs of u and v), and they need 512 clock cycles and 256 clock cycles at most respectively. However, radix-8 modulus inverse algorithm deals with three bits once a clock cycle and only needs 171 clock cycles on average, which means 85 clocks are saved. The details of the algorithm are given by TABLE V below. The focus of the algorithm is to reduce u or v as quickly as possible until one of them is equal to 1, which means the process is over.

Unfortunately, when both of u and v are odd numbers, we can't reduce u or v by simply right shifting. But it's worth noting that when $c+d=3'b000$, we can reduce the $c+d$ with 3 bits by right shifting once a clock cycle and give the result to the maximum of c and d . Furthermore, when $(c-d)=3'b100$ or $(d-c)=3'b100$ we can execute $u\leftarrow(u-v)/4,\ x_1\leftarrow(x_1-x_2)/4$, *if* $u>v$ to guarantee at least one of c and d is reduced by more than two bits. We give all the cases when c and d are odd numbers in TABLEVI. As we know that deciding the cases that $c+d=3'b000$ and $\|c-d\|=3'b100$ is an easy task, so

TABLE V
FAST RADIX-8 MODULUS INVERSE ALGORITHM

Radix-8 Modulus Inverse Algorithm on *GF(p)*

```
Input: prime $p, a \in [1, p-1]$
Output: $a^{-1} \bmod p$
1. $u \leftarrow a, v \leftarrow p, x_1 \leftarrow 1, x_2 \leftarrow 0$
2. if $v > 0$, repeatedly execute
  2.1 $c = u[2:0], d = v[2:0]$
  2.2 if $c = 3'b000$, execute
      $u \leftarrow u/8,\ x_1 \leftarrow x_1/8$
  2.3 else if $d = 3'b000$, execute
      $v \leftarrow v/8,\ x_2 \leftarrow x_2/8$
  2.4 else if $c[1:0] = 2'b00$, execute
      $u \leftarrow u/4,\ x_1 \leftarrow x_1/4$
  2.5 else if $d[1:0] = 2'b00$, execute
      $v \leftarrow v/4,\ x_2 \leftarrow x_2/4$
  2.6 else if $c = d$, execute
      2.6.1 if $u > v, u \leftarrow (u-v)/8,\ x_1 \leftarrow (x_1 - x_2)/8$
      2.6.2 else, $v \leftarrow (v-u)/8,\ x_2 \leftarrow (x_2 - x_1)/8$
  2.7 else if $c[1:0] = d[1:0]$, execute
      2.7.1 if $u > v, u \leftarrow (u-v)/4,\ x_1 \leftarrow (x_1 - x_2)/4$
      2.7.2 else, $v \leftarrow (v-u)/4,\ x_2 \leftarrow (x_2 - x_1)/4$
  2.8 else if $c[0] = 0$, execute
      2.8.1 if $u/2 > v,\ u \leftarrow [(u/2) - v]/2, x_1 \leftarrow [(x_1/2) - x_2]/2$
      2.8.2 else, $u \leftarrow [v - (u/2)]/2, x_1 \leftarrow [x_2 - (x_1/2)]/2$
  2.9 else if $d[0] = 0$, execute
      2.9.1 if $v/2 > u, v \leftarrow [(v/2) - u]/2, x_2 \leftarrow [(x_2/2) - x_1]/2$
      2.9.2 else, $v \leftarrow [u - (v/2)]/2, x_2 \leftarrow [x_1 - (x_2/2)]/2$
  2.10 else if $c + d = 3'b000$
      $\max[u, v] \leftarrow (u+v)/8, x_1/x_2 \leftarrow (x_1 + x_2)/8$
  2.11 else if $u > v$, execute
      2.11.1 if $(c-d) \,||\, (d-c) = 3'b100$
          $u \leftarrow (u-v)/4,\ x_1 \leftarrow (x_1 - x_2)/4$
      2.11.2 else $u \leftarrow (u+v)/4, x_1 \leftarrow (x_1 + x_2)/4$
  2.12 else,
      2.12.1 if $(c-d) \,||\, (d-c) = 3'b100$
          $v \leftarrow (v-u)/4,\ x_2 \leftarrow (x_2 - x_1)/4$
      2.12.2 else $v \leftarrow (u+v)/4, x_1 \leftarrow (x_1 + x_2)/4$
3. return $x_1$
```

the delay of the ***Inv*** module will not increase so much compared with Radix-2 and Radix-4 inverse algorithm, which ensure the critical path of the whole point multiplication hardware architecture unchanged.

TABLE VI
THE CASES WHEN c AND d ARE ODD NUMBERS ($u > v$)

c	**d**	**operations**	**c**	**d**	**operations**
001	011	"+"then">>2"	101	001	"-"then">>2"
001	101	"-"then">>2"	101	011	"+"then">>3"
001	111	"+"then">>3"	101	111	"+"then">>2"
011	001	"+"then">>2"	111	001	"+"then">>3"
011	101	"+"then">>3"	111	011	"-"then">>2"
011	111	"-"then">>2"	111	101	"+"then">>2"

CONCLUSION

In the hardware implementation, we use the basic 0.13*um* CMOS standard cell library to evaluate the performance of our design. For a clear comparison, we give the clock cycles needed to complete the binary inverse in *GF(p)* by radix-2 and radix-4, as well as radix-8 algorithm proposed in this paper, in TABLE VII below.

TABLE VII
CLOCK CYCLES NEEDED TO COMPLETE INVERSE

	Radix-2	Radix-4	Radix-8
Cycles	512	256	171
Percentage in SM2	15.4%	7.7%	5.1%

Compared with radix-2 inverse algorithm, radix-4 inverse algorithm reduces the number of cycles significantly by 100% on average (256 cycles are saved) and improved the performance of SM2 by 7.7%. Similarly, compared with radix-4 inverse algorithm, radix-8 inverse algorithm reduces the number of cycles significantly by 33.2% on average (85 cycles are saved) and improved the performance of SM2 by 2.6%.

ACKNOWLEDGEMENTS

This work is supported by the National Natural Science Foundation of China (Grants 61472208) and National Key Basic Research Program of China (Grant 2013CB338004).

REFERENCES

[1] N. Kobliz, Elliptic curve cryptosystem, Mathematics of Computation 48 (1987) 203-209.
[2] State Cryptography Administration of China: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves, http://www.oscca.gov.cn/UpFile/2010122214822692.pdf
[3] Tim Guneysu, Christof Paar, Ultra High Performance ECC over NIST Primes on Commercial FPGAs, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 5154, 2008, pp. 62C78.
[4] Zhenwei Zhao, Guoqiang Bai. Exploring the Speed Limit of SM2. 2014 IEEE 3rd International Conference on Cloud Computing and Intelligence Systems. IEEE, 2014. 456-460.
[5] Zhenwei Zhao, Guoqiang Bai. Ultra High-Speed SM2 ASIC Implementation. The 13th IEEE Internatinal Conference on Trust,Security and Privacy in Computig and Communicatins. IEEE, 2014. 182-188.
[6] Patrick Longa, Catherine Gebotys, Efficient Techniques for High-Speed Elliptic Curve Cryptography, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 6225, 2010, pp. 80C94.