2014 IEEE 13th International Conference on Trust, Security and Privacy in Computing and Communications

Ultra High-Speed SM2 ASIC Implementation

Zhenwei Zhao
Department of Computer Science and Technology
Tsinghua University
zhaozw12@mails.tsinghua.edu.cn

Guoqiang Bai
Institute of Microelectronics
Tsinghua University
baigq@mail.tsinghua.edu.cn

***Abstract*—In this paper, we present a high-performance elliptic curve cryptographic architecture over SCA-256 prime field by introducing a one-cycle full-precision multiplier. Based on the multiplier, we give a thorough bottom-up optimization in algorithm level. The performance of the architecture is boosted by the use of a two-stage pipeline scheme, and our pipeline utilization reaches 100%. It takes only 8 cycles to perform point doubling and 12 cycles to perform point addition. Using NAF recoding of scalar k, it takes about 3333 cycles to complete the point multiplication operation. In the hardware evaluation using a 0.13 μm CMOS standard cell library, our high-performance SM2 architecture executes one point multiplication operation in 20.36 μs, which translates into more than 49000 point multiplications per second. To the best of our knowledge, our architecture offers the highest single-core performance over prime fields reported in literature up to now.**

Keywords: Elliptic Curve Cryptography, SM2, Point Multiplication, High Speed, ASIC.

I. Introduction

Elliptic curve cryptography (ECC) was proposed independently by Miller [1] and Koblitz [2] in 1985. Since then a considerable amount of researches [13]–[23] have been performed on the efficient hardware implementation of ECC. Unlike RSA, as there is no subexponential-time algorithm that could solve the discrete logarithm problem (DLP), ECC offers the same level of security with smaller key sizes, which results in faster implementation, lower memory usage and higher throughput rate. These advantages make ECC an attractive alternative to traditional public key cryptosystems (such as RSA [5] and Diffie-Hellman [6]). Furthermore, in recent years, ECC has been broadly accepted and standardized by international standard organizations such as ANSI [7], NIST [8] and IEEE [9].

The State Cryptography Administration (SCA) of China promulgates the national elliptic curve public key cryptography algorithm standard in December 2010, known as SM2. SM2 is China's ECC algorithm defined over a special prime field, it has the potential to be widely used to implement public key cryptosystems such as digital signature, encryption/decryption and key establishment schemes for commercial applications. The EC parameters for SM2 are defined as below(in hexadecimal form):

(1) 256-bit prime field
p_{256} =fffffffe ffffffff ffffffff ffffffff ffffffff 00000000 ffffffff ffffffff

(2) Weierstrass equation is $y^2 = x^3 + ax + b$ where
a =fffffffe ffffffff ffffffff ffffffff ffffffff 00000000 ffffffff fffffffc
b =28e9fa9e 9d9f5e34 4d5a9e4b cf6509a7 f39789f5 15ab8f92 ddbcbd41 4d940e93

(3) Coordinates of base point $G = (G_x, G_y)$ are
G_x =32c4ae2c 1f198119 5f990446 6a39c994 8fe30bbf f2660be1 715a4589 334c74c7
G_y =bc3736a2 f4f6779c 59bdcee3 6b692153 d0a9877c c62a4740 02df32e5 2139f0a0
and order of G is
n =fffffffe ffffffff ffffffff ffffffff 7203df6b 21c6052b 53bbf409 39d54123

The recommended 256-bit prime can be rewritten as $p_{256} = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$. In the following discussion, we will use SCA-256 to represent this special prime. SCA-256 is a pseudo-Mersenne prime that allows for efficient modular reduction, just like NIST-256 prime $(2^{256} - 2^{224} + 2^{192} + 2^{96} - 1)$. Another feature of SM2 is coefficient $a = p - 3$, which produces a faster algorithm for elliptic curve point doubling using Jacobian coordinates. More specific details on SM2 can be found in [10].

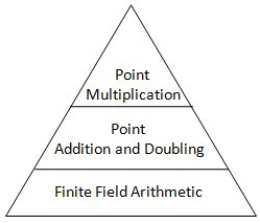

Fig. 1. Hierarchy of operations in point multiplication

The operation that dominates the execution time of elliptic curve cryptography is point multiplication, defined as $kP = \underbrace{P + P + \cdots + P}_{k \text{ times}}$, where k is an integer and P is a point on the elliptic curve. It is computed by a series of point addition and doubling operations of point P. A lot of methods and architectures have been proposed to compute kP [3]. Efficient implementation of point multiplication can be separated into three distinct layers, as shown in Fig. 1.

A wealth of researches regarding high speed hardware implementations of ECC have been conducted. Most of the hardware architectures focus on elliptic curves defined over

978-1-4799-6513-7/14 $31.00 © 2014 IEEE
DOI 10.1109/TrustCom.2014.27

binary field GF(2^m) because it is more hardware-friendly. To date, there appears to be extremely few commercially available ASICs that provide truly high-performance ECC. To the author's knowledge, there is no hardware architecture in the open literature that can perform more than 10000 point multiplications per second on a single-core. In this paper, we conduct a detailed and careful study of the ultra high-speed ECC ASIC implementation over SCA-256 and propose a novel architecture that can realize this goal.

The remainder of this paper is organized as follows. Section II reviews the high-speed ECC hardware implementations over GF(p) reported in the open literature. Section III presents the preliminaries for ECC briefly. Our bottom-up algorithm optimization for point multiplication and the proposed architecture is detailed in Section IV. The SM2 ASIC implementation and performance comparison with other ECC architectures is described in Section V. Finally, we conclude this paper in Section VI.

II. Previous Work

This section briefly summarizes some of the most relevant previous contributions in high-speed ECC implementations.

In [15], the authors propose an elliptic curve cryptographic processor architecture that can support GF(p) and GF(2^m). By optimizing the data bus of dual-field Montgomery multiplier and adopting on-the-fly redundant binary converter, the throughput of EC point multiplication is boosted. Synthesized results show that the high-speed design using 117.5 Kgates with a 64-bit multiplier achieved operation time of 1.21 ms for a 160-bit EC scalar multiplication in GF(p), which translates into approximately 800 point multiplications per second.

In [16], a new architecture for an FPGA-based ultra high performance ECC implementation over NIST primes P-224 and P-256 is presented. The architecture makes intensive use of the DSP blocks in modern FPGAs, which results in very high clock frequency. Synthesized results show that it takes 0.62 ms to complete one point multiplication over NIST prime P-256. They improve the overall system throughput by using a multiple-core implementation. With 16 ECC cores running in parallel, they achieve a throughput of more than 24,000 point multiplications per second for P-256.

In [19], the authors present a systolic array based elliptic curve cryptographic processor for general curves over GF(p). They propose a unified systolic arithmetic unit that efficiently calculates modular addition, subtraction, multiplication and division. By delaying the conditional operations and distributing the registers into individual processing elements, pipeline stalls are avoided. Synthesized in 0.13 μm standard cell technology, the processor requires 1.01 ms to compute a 256-bit prime field point multiplication, this corresponds to nearly 1000 point multiplications per second.

In [24], a hardware-efficient dual field ECC design on arbitrary ECs is proposed. It introduces a single-chip dual-processing-element architecture with pipelining and arithmetic unit integration techniques to improve the hardware speed and circuit utilization. What's more, it exploits the key-independent scheduling with masked intermediate data technique to counteract side channel attacks. Fabricated in 90-nm CMOS technology, the chip runs at 217 MHz and takes about 0.76 ms to perform one 256-bit GF(p) point multiplication, the total throughput is 1315 point multiplications per second.

III. Mathematical Background

A. Elliptic Curve over GF(p)

This section provides a brief introduction to elliptic curve, more information can be found in [3].

A non-supersingular elliptic curve E over GF(p) is often expressed in terms of Weierstrass equation: $y^2 = x^3 + ax + b$, where $a, b \in GF(p)$ with $4a^3 + 27b^2 \neq 0 \pmod p$. A point $P = (x, y)$ is a valid point on the elliptic curve if it fulfills the Weierstrass equation. The set of all these points (x, y) together with the point at infinity ∞ (serving as identity of the group) and the operation of "addition", form an abelian group. Let $P_1 = (X_1, Y_1, Z_1)$ and $P_2 = (X_2, Y_2, Z_2)$, the addition is defined as $P_3 = P_1 + P_2$, where $P_3 = (X_3, Y_3, Z_3)$. When $P_1 = P_2$, we have the point doubling formulas, otherwise we have the point addition formulas. The addition formulas vary for affine and projective coordinates. To avoid time-consuming inversion operation, we focus on projective coordinates representation. Because Jacobian coordinates yield the fastest point doubling and mixed Jacobian-affine coordinates yield the fastest point addition [3], we adopt these two coordinates in our design.

Point doubling formulas for computing $2P = (X_3, Y_3, Z_3)$ in Jacobian coordinates are:

$$\begin{aligned} X_3 &= (3X_1^2 + aZ_1^4)^2 - 8X_1Y_1^2 \\ Y_3 &= (3X_1^2 + aZ_1^4)(4X_1Y_1^2 - X_3) - 8Y_1^4 \\ Z_3 &= 2Y_1Z_1 \end{aligned} \tag{1}$$

Point addition formulas using mixed Jacobian-affine coordinates are:

$$\begin{aligned} X_3 &= (Y_2Z_1^3 - Y_1)^2 - (X_2Z_1^2 - X_1)^2(X_1 + X_2Z_1^2) \\ Y_3 &= (Y_2Z_1^3 - Y_1)(X_1(X_2Z_1^2 - X_1)^2 - X_3) - Y_1(X_2Z_1^2 - X_1)^3 \\ Z_3 &= (X_2Z_1^2 - X_1)Z_1 \end{aligned} \tag{2}$$

B. Non-adjacent Form (NAF)

The total EC operation counts for point multiplication are closed related to the representation of scalar k. When k is in binary form, the expected number of operations using double-and-add algorithm [3] is approximately $\frac{m}{2}A + mD$, where m is the length of scalar k, A and D denote point addition and doubling operation respectively. As computing the additive inverse of a given point is almost for free, subtraction of point on an elliptic curve is just as efficient as addition, this fact leads to the use of NAF representation.

A non-adjacent form of a positive integer k is an expression $k = \sum_{i=0}^{l-1} k_i 2^i$, where $k_i \in \{0, \pm 1\}, k_{l-1} \neq 0$. Any integer has only one non-adjacent form where "1" and"-1" are never adjacent. NAF(k) has the fewest nonzero bits that is expected to be one-third of bit length. If scalar k is in NAF

representation, the expected number of operations in point multiplication calculation reduces to $\frac{m}{3}A + mD$. NAF(k) can be efficiently computed using Algorithm 1.

Algorithm 1 Computing the NAF of a positive integer

Require: A positive integer k
Ensure: NAF(k)
1: $i \leftarrow 0$
2: **while** $k \geq 1$ **do**
3: If k is odd then: $k_i \leftarrow 2 - (k \bmod 4)$,$k \leftarrow k - k_i$
4: Else: $k_i \leftarrow 0$
5: $k \leftarrow k/2, i \leftarrow i + 1$
6: **end while**
7: Return $(k_{i-1}, k_{i-2}, \cdots, k_1, k_0)$

IV. SM2 Architecture

In this section, we will give a thorough bottom-up algorithm optimization for SM2 on the basis of Fig. 1 in Section I.

A. Optimizations for Finite Field Arithmetic

The modular arithmetic operations in the finite field are invoked many times by the upper elliptic curve operation, thus the efficient realization of these operations has a significant impact on the performance of an elliptic curve cryptosystem. The basic modular operations include modular addition/subtraction, modular multiplication and modular inversion.

Algorithm 2 Fast reduction with $p_{256} = 2^{256} - 2^{224} - 2^{96} + 2^{64} - 1$

Require: An integer $c = (c_{15}, \cdots, c_2, c_1, c_0)$ in base 2^{32}, $0 \leq c < p_{256}^2$
Ensure: $c \pmod{p_{256}}$
1: Define 256-bit integers
$s_1 = (c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$,
$s_2 = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, 0, c_9, c_8)$,
$s_3 = (c_{14}, 0, c_{15}, c_{14}, c_{13}, 0, c_{14}, c_{13})$,
$s_4 = (c_{13}, 0, 0, 0, 0, 0, c_{15}, c_{14})$,
$s_5 = (c_{12}, 0, 0, 0, 0, 0, 0, c_{15})$,
$s_6 = (c_{11}, c_{11}, c_{10}, c_{15}, c_{14}, 0, c_{13}, c_{12})$,
$s_7 = (c_{10}, c_{15}, c_{14}, c_{13}, c_{12}, 0, c_{11}, c_{10})$,
$s_8 = (c_9, 0, 0, c_9, c_8, 0, c_{10}, c_9)$,
$s_9 = (c_8, 0, 0, 0, c_{15}, 0, c_{12}, c_{11})$,
$s_{10} = (c_{15}, 0, 0, 0, 0, 0, 0, 0)$,
$s_{11} = (0, 0, 0, 0, 0, c_{14}, 0, 0)$,
$s_{12} = (0, 0, 0, 0, 0, c_{13}, 0, 0)$,
$s_{13} = (0, 0, 0, 0, 0, c_9, 0, 0)$,
$s_{14} = (0, 0, 0, 0, 0, c_8, 0, 0)$
2: Return $(s_1 + s_2 + 2s_3 + 2s_4 + 2s_5 + s_6 + s_7 + s_8 + s_9 + 2s_{10} - s_{11} - s_{12} - s_{13} - s_{14}) \bmod p_{256}$

Fast Reduction Scheme. The modular operations defined over a prime field GF(p) have the feature that all these operations need to reduce the regular arithmetic results to the domain of the given field before they can be used for another calculation, so is the modular multiplication. The SCA-256 in SM2 has the property that it can be written as the sum or difference of a small number of powers of 2, only shifts and additions are needed to perform the reduction. Let c be an 512-bit integer with $0 \leq c < p_{256}^2$ and the base-2^{32} representation of c can be denoted as $c = c_{15}2^{480} + c_{14}2^{448} + ... + c_1 2^{32} + c_0$, where $c_i \in [0, 2^{32} - 1]$. The higher powers of 2 in c can be reduced using the congruences as below:

$$\begin{aligned} 2^{256} &\equiv 2^{224} + 2^{96} - 2^{64} + 1 (mod\ p_{256}) \\ 2^{288} &\equiv 2^{224} + 2^{128} - 2^{64} + 2^{32} + 1 (mod\ p_{256}) \\ &\cdots \\ 2^{480} &\equiv 3 \times 2^{224} + 2^{192} + 2 \times 2^{160} + 2^{128} + 2^{96} + 2 \times 2^{32} + 2 (mod\ p_{256}) \end{aligned} \tag{3}$$

Thus, $c \pmod{p_{256}}$ can be calculated by adding eighteen 256-bit integers and reduce the sum to $[0, p_{256} - 1]$ by repeatedly subtracting p. The details of the fast reduction algorithm is described in Algorithm 2. Compared with the commonly used Montgomery-based modular multiplication scheme, fewer additions and shifts are needed to complete the reduction, which highly improves the overall performance.

Full-Precision Multiplier. Most traditional high-speed ECC implementations use a multiplier-based architecture. However, due to limitations of multiplier size, the long operands in ECC algorithm need to be split into small words and processed in many cycles, which is not conducive to high-speed ECC implementation. Table I summarizes performance difference between the full- and multi-precision 256-bit multiplier [20].

TABLE I
PERFORMANCE COMPARISON BETWEEN DIFFERENT 256-BIT MULTIPLIERS

Basic multiplier size	Cycles	Critical path delay(ns)	Total delay(ns)
16	9	2.84	25.56
32	7	3.52	24.64
64	5	4.2	21
128	3	4.97	14.91
256	1	5.65	5.65

We can tell from Table I that a one-cycle 256-bit multiplier will significantly accelerate the speed of modular multiplication, which is the most time-consuming operation in our architecture.

Algorithm 3 Modular multiplication in $GF(p)$

Require: $p, A, B \in [0, p-1]$
Ensure: $R = A \times B \pmod{p}$
1: $T = A \times B$
2: $R = T \pmod{p}$
3: Return R

Based on the full-precision multiplier and the fast reduction unit, modular multiplication is achieved by a regular multiplication followed by a modular reduction. Algorithm 3 gives the new modular multiplication algorithm. Besides, we do not distinguish between multiplication and squaring in our design.

Modular Inversion. We adopt a radix-4 binary inversion algorithm to speed up the time-consuming modular inversion operation, the details is shown in Algorithm 4.

Algorithm 4 Modular inversion in GF(p)

Require: $p, a \in [1, p-1]$
Ensure: $a^{-1} \pmod p$
1: $u \leftarrow a, v \leftarrow p, x_1 \leftarrow 1, x_2 \leftarrow 2$
2: **while** $v > 0$ **do**
3: $c = u[1:0], d = v[1:0]$
4: **if** $c = 0$ **then**
5: $u \leftarrow \frac{u}{4}, x_1 \leftarrow \frac{x_1}{4} \pmod p$
6: **else if** $d = 0$ **then**
7: $v \leftarrow \frac{v}{4}, x2 \leftarrow \frac{x_2}{4} \pmod p$
8: **else if** $c = d$ **then**
9: **if** $u > v$ **then**
10: $u \leftarrow \frac{u-v}{4}, x_1 \leftarrow \frac{x_1 - x_2}{4} \pmod p$
11: **else**
12: $v \leftarrow \frac{v-u}{4}, x_2 \leftarrow \frac{x_2 - x_1}{4} \pmod p$
13: **end if**
14: **else if** $c = 2$ **then**
15: **if** $u/2 > v$ **then**
16: $u \leftarrow \frac{\frac{u}{2} - v}{2}, x_1 \leftarrow \frac{\frac{x_1}{2} - x2}{2} \pmod p$
17: **else**
18: $u \leftarrow \frac{u}{2}, x_1 \leftarrow \frac{x_1}{2} \pmod p$,
$v \leftarrow \frac{v - \frac{u}{2}}{2}, x_2 \leftarrow \frac{x_2 - \frac{x_1}{2}}{2} \pmod p$
19: **end if**
20: **else if** $d = 2$ **then**
21: **if** $u > v/2$ **then**
22: $v \leftarrow \frac{v}{2}, x_2 \leftarrow \frac{x_2}{2} \pmod p$,
$u \leftarrow \frac{u - \frac{v}{2}}{2}, x_1 \leftarrow \frac{x_1 - \frac{x_2}{2}}{2} \pmod p$
23: **else**
24: $v \leftarrow \frac{\frac{v}{2} - u}{2}, x_2 \leftarrow \frac{\frac{x_2}{2} - x_1}{2} \pmod p$
25: **end if**
26: **else**
27: **if** $u > v$ **then**
28: $u \leftarrow \frac{u-v}{2}, x_1 \leftarrow \frac{x_1 - x_2}{2} \pmod p$
29: **else**
30: $v \leftarrow \frac{v-u}{2}, x_2 \leftarrow \frac{x_2 - x_1}{2} \pmod p$
31: **end if**
32: **end if**
33: **end while**
34: Return x_1

The algorithm maintains the invariants:

$$\begin{aligned} ax_1 &\equiv u (mod\ p) \\ ax_2 &\equiv v (mod\ p) \end{aligned} \tag{4}$$

The operands u, v, x_1, x_2 are initialized as $a, p, 1, 0$ respectively. In each iteration, the values of u and v are updated according to their last two bits. Meanwhile, x_1 and x_2 are changed correspondingly to make sure the invariants still hold. The iteration will terminate when $u = 1$, and x_1 will be equal to $a^{-1} \pmod p$.

In Algorithm 4, halving and quartering of u and v can be performed by simple right-shifting operation, because their least significant one or two bits are always zero. However, this is not true for x_1 and x_2 whose least significant bits are not determined. The modular halving and quartering formulas are defined as follows:

$$\frac{x}{2}(mod\ p) = \begin{cases} \frac{x}{2} & if\ x\ is\ even \\ \frac{x+p}{2} & otherwise \end{cases} \tag{5}$$

$$\frac{x}{4}(mod\ p) = \begin{cases} \frac{x}{4} & if\ x\ mod\ 4 = 0 \\ \frac{x+2p}{4} & if\ x\ mod\ 4 = 2 \\ \frac{x+3p}{4} & if\ (x+p)\ mod\ 4 = 2 \\ \frac{x+p}{4} & otherwise \end{cases} \tag{6}$$

The maximum number of iterations in Algorithm 4 is k, with k the length of modulus p, and each iteration can be done in one cycle. Hence, the inversion latency is at most k cycles.

Modular Addition/Subtraction. Just as the multiplier, we use a full-precision adder to perform the modular addition/subtraction operation according to Algorithm 5.

Modular addition of two integers $A, B \in [0, p-1]$ is done by first calculating the sum $S_0 = A + B$ using the adder, then the difference $\{C_1, S_1\} = S_0 - p$. If C_1 is set, then the sum $A+B$ exceeds $p-1$, a modular reduction is necessary to guarantee that the final result is in the range $[0, p-1]$. The subtraction operation is usually achieved by adding twos complement of the subtrahend. Therefore addition and subtraction can be realized by using the same circuit.

In our final design, modular addition/subtraction is merged into the modular reduction operation for the sake of balancing paths and improving performance, as will discussed in Section IV-B.

Algorithm 5 Modular addtion and subtraction in $GF(p)$

Require: $p, A, B \in [0, p-1]$
Ensure: $R = (A + B) \bmod p$
1: $\{C_0, S_0\} = A + B$
2: $\{C_1, S_1\} = S_0 - p$
3: **if** $C_1 = 1$ **then**
4: $R = S_1$
5: **else**
6: $R = S_0$
7: **end if**
8: Return R

Require: $p, A, B \in [0, p-1]$
Ensure: $R = (A - B) \bmod p$
1: $\{C_0, S_0\} = A - B$
2: $\{C_1, S_1\} = S_0 + p$
3: **if** $C_0 = 1$ **then**
4: $R = S_0$
5: **else**
6: $R = S_1$
7: **end if**
8: Return R

B. Optimizations for Point Addition and Doubling

It is important to note that an efficient implementation of modular arithmetic operations in finite field does not necessary yield a high-speed point multiplication, algorithm optimization at the point arithmetic layer is also very important.

Point Addition and Doubling Formulas. The formulas of point addition and doubling operations in our design are shown in (1) and (2), which are eventually completed by the

use of the modular units in the finite filed. Compared with modular multiplication, the time needed to perform modular addition/subtraction can be negligible. However, there might be data dependencies between contiguous field operations in the formulas. That means, in some cases, addition/subtraction operation must wait until multiplication finishes. Extra multiplication cycle is needed to perform these less time-consuming operations and it has an adverse effect on the performance of point multiplication.

Algorithm 6 Point doubling ($y^2 = x^3 - 3x + b$)

Require: $P_1 = (X_1, Y_1, Z_1)$ in Jacobian coordinates
Ensure: $P_3 = 2P_1 = (X_3, Y_3, Z_3)$ in Jacobian coordinates
1: $T_1 \leftarrow Z_1 Z_1$
2: $T_2 \leftarrow Y_1 Y_1$, $T_3 \leftarrow X_1 + T_1$, $T_4 \leftarrow X_1 - T_1$
3: $T_1 \leftarrow T_3 T_4$
4: $T_3 \leftarrow Y_1 Z_1$, $T_4 \leftarrow 8T_2$
5: $T_5 \leftarrow X_1 T_4$, $T_1 \leftarrow 3T_1$
6: $T_3 \leftarrow T_1 T_1$, $Z_3 \leftarrow T_3 + T_3$
7: $T_2 \leftarrow T_2 T_4$, $X_3 \leftarrow T_3 - T_5$, $T_4 \leftarrow 1.5T_5 - T_3$
8: $T_1 \leftarrow T_1 T_4$
9: $Y_3 \leftarrow T_1 - T_2$
10: Return (X_3, Y_3, Z_3)

In the best case, addition/subtraction required for EC operations is fully in parallel with multiplication operation and the multiplier is kept busy at all times. A detailed analysis is conducted on the execution order of the basic field operations in point addition and doubling formulas. Fortunately, we have found the optimal scheduling scheme, Algorithm 6 and Algorithm 7 give the optimization results for point doubling and addition procedures respectively.

Algorithm 7 Point addition ($y^2 = x^3 - 3x + b$)

Require: $P_1 = (X_1, Y_1, Z_1)$ in Jacobian coordinates,$P_2 = (x_2, y_2)$ in affine coordinates
Ensure: $P_3 = P_1 + P_2 = (X_3, Y_3, Z_3)$ in Jacobian coordinates
1: $T_1 \leftarrow Z_1 Z_1$
2: $T_2 \leftarrow y_2 Z_1$
3: $T_3 \leftarrow x_2 T_1$
4: $T_1 \leftarrow T_1 T_2$, $T_2 \leftarrow T_3 - X_1$, $T_3 \leftarrow T_3 + X_1$
5: $T_4 \leftarrow T_2 T_2$, $T_1 \leftarrow T_1 - Y_1$
6: $Z_3 \leftarrow Z_1 T_2$
7: $T_2 \leftarrow T_2 T_4$
8: $T_3 \leftarrow T_3 T_4$
9: $T_5 \leftarrow T_1 T_1$
10: $T_4 \leftarrow X_1 T_4$, $X_3 \leftarrow T_5 - T_3$
11: $T_2 \leftarrow Y_1 T_2$, $T_3 \leftarrow T_4 - X_3$
12: $T_1 \leftarrow T_1 T_3$
13: $Y_3 \leftarrow T_1 - T_2$
14: Return (X_3, Y_3, Z_3)

Pipeline Scheme. In this section, we propose a two-stage pipeline scheme to increase the clock frequency and improve the overall performance.

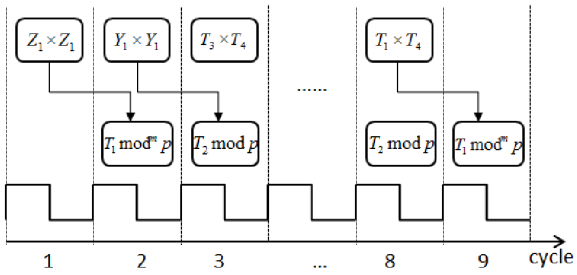

Fig. 2. Pipeline scheduling scheme for point doubling

A key observation is that the calculations of regular multiplication and modular reduction are two separate processes, while reduction unit is in progress, the multiplier is totally idle. By splitting modular multiplication into two stages: regular multiplication stage and modular reduction stage, the multiplier is fully utilized. Another observation is that the delay sum of fast reduction unit and modular adder/substractor is less than that of full-precision multiplier, so we merge modular addition/substraction into multiplicative reduction to balance paths and avoid pipeline stalls. Based on these optimizations, Fig. 2 gives a detailed pipeline scheduling scheme for point doubling, where "$\times$" denotes 256-bit regular multiplication, "$mod\ p$" means regular reduction using Algorithm 2 and "$mod^m\ p$" represents reduction mixed with modular addition/substraction. Similar technique can also be applied to point addition procedure in Algorithm 7.

With this spilt, the reduction operation is totally in parallel with consecutive multiplication operation, it seems as if all these modular multiplications are regular multiplications and no reduction is needed. Moreover, there is no pipeline stall and our pipeline scheme utilization reaches 100%.

C. Optimizations for Point Multiplication

In this section, we will discuss the algorithm used for computation of point multiplication.

Algorithm 8 On-the-fly NAF for point multiplication

Require: An integer $k, P = (x, y) \in E(F_q)$
Ensure: $Q = kP$
1: calculate $3k = \sum_{i=0}^{l} h_i 2^i$, where h_l is 1
2: calculate $k = \sum_{i=0}^{l} k_i 2^i$
3: $Q \leftarrow P$
4: **for** i from $l-1$ downto 1 **do**
5: $Q \leftarrow 2Q$
6: If h_i=1 and k_i=0 then $Q \leftarrow Q + P$
7: If h_i=0 and k_i=1 then $Q \leftarrow Q - P$
8: **end for**
9: Return Q

As discussed in section III-B, NAF representation of scalar k has the merits of reducing point multiplication time complexity from $\frac{m}{2}A + mD$ to $\frac{m}{3}A + mD$. Algorithm 1

describes the basic method to convert an integer into NAF. However, it is calculated from the LSB side while the addition-and-substraction point multiplication method [3] for NAF requires bits from the MSB side. If we adopt Algorithm 1, a lot of cycles are wasted on the calculation of NAF(k). Therefore we abandon this algorithm and utilize the method that precalculates $3k$, with the cost of extra register resources. Algorithm 8 gives a detailed description of this method.

TABLE II
CYCLE NUMBER FOR EC OPERATIONS

operation	cycles
modular addition	1
modular substraction	1
modular multiplication	2
modular inversion	256
point addition	12
point doubling	8
point multiplication	3333

Up to now, we have completed the algorithm optimizations in every layer of the point multiplication scheme. Table II summarizes the cycle number to perform basic EC operations after these optimizations.

Our new architecture only needs 8 cycles to perform point doubling and 12 cycles to perform point addition. Using NAF representation of scalar k, it takes about 3333 cycles to complete one point multiplication operation.

D. SM2 Architecture

In the sections above, we have described a complete scheme to perform point multiplication in SM2. Fig. 3 shows a block diagram of this architecture. There are mainly three components in the diagram:

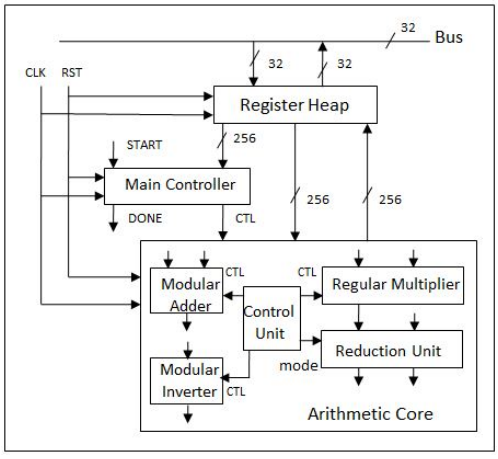

Fig. 3. SM2 Architecture

Arithmetic Core (AC): AC is composed of modular adder, regular multiplier, reduction unit and performs the basic field operations under the control of the Control Unit inside. The Control Unit is instructed directly by Main Controller.

Main Controller (MC): MC instructs AC to perform appropriate elliptic curve operations according to the bit of scalar k, and sets DONE signal when the calculation is over to interact with the host system.

Register Heap (RH): RH stores coordinates of the points, EC parameters and intermediate variables generated in the computing process.

The proposed architecture demonstrates superior performance for point multiplication calculation and it has only a single instance of the arithmetic units. Furthermore, our architecture can be easily reconfigured to support the NIST-256 prime. We just need to modify the reduction unit to adapt to NIST-256 reduction algorithm and the overall performance is not affected at all.

V. IMPLEMENTATION

A. Implementation Results

The architecture described in Section IV-D has been captured and verified in Verilog-HDL. In the hardware evaluation stage, we synthesize our design using 0.13 μm CMOS standard cell library. Results show that the critical path delay is 6.11 ns, which may result in a clock frequency of 163.7 MHz. The total area of the circuit is 659K gates, where one gate unit is a 2-way NAND.

It takes about 3333 cycles to complete the point multiplication operation, so our high-performance SM2 architecture executes one point multiplication in 20.36 μs, which translates into more than 49000 point multiplications per second. To the best of our knowledge, this architecture offers the fastest performance for ECC computation over prime fields with up to 256-bit security.

TABLE III
PERFORMANCE COMPARISON WITH OTHER ARCHITECTURES

paper	our work	[15]	[19]	[25]	[16]
curve	GF(p) SCA-256	GF(p) 256-bit	GF(p 256-bit	GF(p) 256-bit	GF(p) NIST-256
platform	0.13 μm	0.13 μm	0.13 μm	90 nm	XC4VSX55
freq.(MHz)	163.7	137.7	556	250	375
area	659K	120K	122K	122K	24574 LS /512 DSP
cycles	3.3K	340K	562K	193K	303K
time(μs)	20.36	2680	1010	770	808
AT$^{\alpha}$	1	23.97	9.18	10.14$^{\beta}$	-
kP/s	49105	373	990	1298	19760$^{\gamma}$

$^{\alpha}$ AT = area-time product.
$^{\beta}$ Normalization factor is 1.45 (130-nm/90-nm).
$^{\gamma}$ Obtained by 16 ECC cores running in parallel.

B. Performance Comparison and Analysis

We summarize previously published hardware results targeting 256-bit prime field and a detailed performance comparison with other implementations is presented in Table III. It should be noted that a fair comparison between these results is

infeasible, as the EC parameters, design methodologies and application areas are not exactly the same.

As shown in Table III, among all the hardware implementations, our architecture is the only one that can perform more than 10000 point multiplications per second on a single core. The second best time is obtained in [16], which is also highly optimized for NIST-256 prime field. However, the high performance is achieved by 16 ECC cores running in parallel and their single core speed is 1614 kP/s , which is 1/30th of our speed.

Reference [25] is the fastest ASIC implementation we could find that supports 256-bit prime field in the open literature, the result is post-layout simulation using 90 nm CMOS technology. Comparison results show that our design area is 5.4 times larger while our speed is 37.8 times faster.

Compared with our design, reference [19] runs at a higher frequency due to the finely pipelined carry chain of systolic array. However, their cycle number is 170 times larger and in total our processor is 49.6 times faster.

We also compare the performance difference between a full- and multi-precision multiplier ECC implementation. Based on a 64-bit multiplier, reference [15] can achieve operation time of 2.68 ms with cycle number as large as 340K. Our 256-bit multiplier architecture only needs 3.3K cycles to perform point multiplication and the speed is 131.6 times faster.

The AT row in Table III shows that our design outperforms other ECC designs in terms of area-time product. We can conclude from these figures that the increase in area is not the only factor that leads to the ultra high performance, the optimizations we have done also play a key role. However, speed is gained at the expense of flexibility. Our architecture is specific to SM2 algorithm and all algorithms are optimized for that particular prime. It is difficult to change the hardware configurations to support other ECC algorithms. What's more, in our design principle, performance is of utmost importance and we don't take side channel protection into consideration, which is a real threat for modern security integrated circuit.

VI. Conclusion

In this paper, we present an ultra high-performance point multiplication scheme for SM2. For the first time, we introduce a one-cycle full-precision multiplier to accelerate the speed of modular multiplication. Based on the multiplier, we have conducted a thorough bottom-up optimization to all layers of point multiplication in algorithm level and then map them into hardware architecture. A two-stage pipeline is applied to increase clock frequency and boost performance of the scheme. Synthesized results show that we could achieve a point multiplication time of 20.36 μs, so over 49000 point multiplications per second can be performed.

In our future work, we will focus on exploring and incorporating state-of-the-art techniques into our architecture to counteract side channel attacks.

References

[1] V. Miller, Uses of elliptic curves in cryptography, in: Cryptology Conference (CRYPTO), LNCS 218, 1985, pp.417C426.

[2] N. Koblitz, Elliptic curve cryptosystem, Mathematics of Computation 48 (1987) 203C209.

[3] D. Hankerson, A. Menezes, and S. Vanstone, Guide to Elliptic Curve Cryptography, Springer, New York, 2004.

[4] "IEEE P1363 Draft Version D13, Standard for Public-Key Cryptography, Draft Standard," Nov. 1999.

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

[6] W. Diffie and M. Hellman, New directions in cryptography, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp. 644C654.

[7] ANSI X9.63, Public Key Cryptography for the Financial Services Industry: Elliptic Curve Key Agreement and Key Transport Protocols, October 2000.

[8] National Institute of Standards and Technology, Digital Signature Standard, FIPS Publication 186-2, February 2000.

[9] Institute of Electrical and Electronic Engineers, NY, P1363 standard specifications for public key cryptography, 2000.

[10] State Cryptography Administration of China: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves, http://www.oscca.gov.cn/UpFile/2010122214822692.pdf

[11] P. Montgomery, Modular Multiplication without Trial Division. Mathematics of Computation, vol. 44, no. 170, April 1985, pp. 519-521.

[12] A. J. Menezes, P. C. van Oorschot, and S. A. Vanstone,Handbook of Applied Cryptography. CRC Press, 1997.

[13] de Dormale, G.M., Quisquater, J.-J.: High-speed hardware implementations of elliptic curve cryptography: A survey. J. Syst. Archit. 53(2-3), 72C84 (2007).

[14] S.B. Ors, L. Batina, B. Preneel, J. Vandewalle, Hardware implementation of an elliptic curve processor over GF(p), in: Application-Specific Systems, Architectures, and Processors(ASAP), 2003, pp. 433C443.

[15] A. Satoh, K. Takano, A scalable dual-field elliptic curve cryptographic processor, IEEE Transactions on Computers.52(4), 449C460(2003).

[16] 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.

[17] Patrick Longa, Catherine Gebotys, Efficient Techniques for High-Speed Elliptic Curve Cryptography, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 6225, 2010, pp. 80C94.

[18] Kendall Ananyi, Hamad Alrimeih, and Daler Rakhmatov, Flexible Hardware Processor for Elliptic Curve Cryptography Over NIST Prime Fields, in: IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, 17(8), 1099- 1112(2009).

[19] Chen G, Bai G, Chen H. A high-performance elliptic curve cryptographic processor for general curves over GF (p) based on a systolic arithmetic unit[J]. Circuits and Systems II: Express Briefs, IEEE Transactions on, 2007, 54(5): 412-416.

[20] Ciaran J. McIvor, Mire McLoone, Hardware Elliptic Curve Cryptographic Processor Over GF(p), in: IEEE TRANSACTIONS ON CIRCUITS AND SYSTEMS. 53(9), 1946-1957(2006).

[21] Pradeep Kumar Mishra, Pipelined Computation of Scalar Multiplication in Elliptic Curve Cryptosystems, in: Cryptographic Hardware and Embedded Systems (CHES), LNCS 3156, 2004, pp. 328C342.

[22] William N. Chelton, Mohammed Benaissa, Fast Elliptic Curve Cryptography on FPGA, in: IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, 16(2), 198-205 (2008).

[23] B. Ansari, M. Anwar Hasan, High performance architecture of elliptic curve scalar multiplication,Tech. Report CACR2006-01, 2006. Available from: http://www.cacr.math.uwaterloo.ca/techreports/2006/cacr2006-01.pdf

[24] Lee J W, Chung S C, Chang H C, et al. Efficient Power-Analysis-Resistant Dual-Field Elliptic Curve Cryptographic Processor Using Heterogeneous Dual-Processing-Element Architecture[J]. Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, 2014, 22(1): 49-61.

[25] Chen Y L, Lee J W, Liu P C, et al. A dual-field elliptic curve cryptographic processor with a radix-4 unified division unit[C]//Circuits and Systems (ISCAS), 2011 IEEE International Symposium on. IEEE, 2011: 713-716.