FourQ on ASIC: Breaking Speed Records for Elliptic Curve Scalar Multiplication

Hiromitsu Awano* and Makoto Ikeda†
VLSI Design and Education Center, The University of Tokyo
Hongo 7-3-1, Bunkyo-ku, Tokyo, Japan, 113–8656
Email: *awano@vdec.u-tokyo.ac.jp, †ikeda@silicon.u-tokyo.ac.jp

***Abstract*—An ASIC cryptoprocessor for scalar multiplication (SM) on FourQ is proposed. By exploiting Karatsuba multiplication and lazy reduction techniques, the arithmetic units of the proposed processor are tailored for operations over quadratic extension field ($\mathbb{F}_{p^2}$). We also propose an automated instruction scheduling methodology based on a combinatorial optimization solver to fully exploit the available instruction-level parallelism. With the proposed processor fabricated by using a 65 nm silicon-on-thin-box (SOTB) CMOS process, we demonstrate that an SM can be computed in 10.1 μs when a typical operating voltage of 1.20 V is applied, which corresponds to 3.66× acceleration compared to the conventional P-256 curve SM accelerator implemented on an ASIC platform and is the fastest ever reported. We also demonstrate that by lowering the supply voltage down to 0.32 V, the lowest ever reported energy consumption of 0.327 μJ/SM is achieved.**

I. Introduction

Ever increasing traffic densities in many countries causes increasing travel times, fuel consumption and carbon dioxide emission. A report showed that the economic losses caused by the traffic congestion have reached 200 billion euro in Europe and 101 billion dollar in US [1]. At the same time, the infrastructure in most cities have already reached the limit at which further urban development may cost prohibitively expensive or be even impossible. Hence, intelligence transportation systems (ITS) attract increasing attention to make the traffic flow itself as more efficient by exploiting a variety of communication, computing, control and sensing technologies. A good example can be found in traffic management systems (TMSs), in which the intersection-approach traffics are detected by wireless sensor devices deployed in the road environment to adaptively control the traffic light intervals so that both the traffics and pedestrian-crossing flow more smoothly.

It is clear that these kind of convenience may be threatened by various security risks; e.g., an attacker may send a faked massage to impersonate a priority vehicle, which disturbs the traffic flow. Authenticating messages are, therefore, indispensable for ITS, and hence use of digital signature algorithms (DSAs) is recommended in European ITS security standard [2].

The early digital signature scheme is based on RSA, whose security is based on the practical difficulty of the factorization of very large number composed of the product of two large prime numbers [3]. However, due to the rapid improvement of the computing capability, the recommended key length for RSA has continued to increase. As of 2016, the NSA suggest the use of 3072 bit or larger keys, which makes implementation of the RSA algorithm as quite challenging especially in embedded systems.

To reduce the key size without degrading the security level, elliptic curve digital signature algorithms (ECDSAs) have been developed. In comparison to RSA, ECDSA exploits a scalar multiplication, the repetitive additions of a point along an elliptic curve, to produce a one-way function. ECDSA is known to achieve the same security strength with only a 1/10 of key length used in RSA [4], e.g., a 256 bit elliptic curve public key should provide comparable security to a 3072 bit RSA public key, which drastically reduce the key storage and transmission cost and is suitable for embedded applications.

In spite of the aggressive research on ECDSAs, their application to ITS is not straightforward due to the severe throughput and latency requirements. Assume that a typical dense traffic situation where an urban road with six lanes are fulled with vehicles and that each vehicle transmits/receive a message to/from the other vehicles or road infrastructures, it is clear that the situation easily leads to a large flood of messages to be verified. For example, Knežević et al. showed that the number of messages will reach 1000 per second when the channel bandwidth, utilization, and the message length are 6 Mb, 33%, and 256 bytes, respectively [5]. However, due to the rapid evolution of the wireless communication technology, the network bandwidth is expected to reach 100 Mb/s for metropolitan areas [6], and hence further improvement in DSA throughput is practically important to fully exploit the available network bandwidth.

In this paper, we propose an application specific integrated circuit (ASIC) cryptoprocessor for scalar multiplications (SMs) over FourQ to further increase the DSA throughput. FourQ is a hardware-friendly 128 bit secure elliptic curve proposed by Castello and Longa [7]. Due to the adoption of a four-dimensional decomposition [8] combined with the twisted Edwards explicit formulas and Mersenne prime $p = 2^{127} - 1$, FourQ achieves 5× speed-up compared to the standardized NIST P-256 curve and approximately 2× faster than the Curve25519 which is also known to be efficient [9].

Based on the profiling of FourQ's SM algorithm, which reveals that $\mathbb{F}_{p^2}$ multiplications accounts for approximately 57% of total arithmetic operations, we choose to design an arithmetic units tailored for $\mathbb{F}_{p^2}$ multiplications, with which a single $\mathbb{F}_{p^2}$ multiplication can be computed per cycle. We further proposes an automated instruction scheduling flow based on a combinatorial optimization solver. Since an SM is composed of thousands of atomic operations on $\mathbb{F}_{p^2}$, the instruction scheduling is quite important to fully exploit the available hardware resource and to minimize the latency. In conventional cryptoprocessor designs such as in [10], the

978-3-9819263-2-3/DATE19/©2019 EDAA

instruction scheduling is optimized manually, which is prone to errors and also leads to increased turn-around-time. In our design flow, the algorithm for computing SM over $\mathrm{Four}\mathbb{Q}$ is written by using a Python script, whose execution trace is recorded to extract the execution order of atomic operations on $\mathbb{F}_{p^2}$. Finally, based on the extracted execution order, a job-shop optimization problem is formulated, which is solved by using a commercial optimization solver to generate the finite state machine (FSM) controller.

The proposed accelerator is designed and fabricated by using 65 nm silicon-on-thin-box (SOTB) CMOS process. The chip measurement result shows that the fabricated chip successfully computes an SM in 10.1 μs when a typical operation voltage of 1.2 V is applied, which is 15.5$\times$ speed-up compared to the $\mathrm{Four}\mathbb{Q}$ implemented on an FPGA platform [10] and 3.66$\times$ acceleration compared to the fastest ever reported elliptic curve accelerator on an ASIC platform [5]. Further, by reducing the operating voltage down to 0.32 V, the energy for performing an SM can be minimized to 0.327 μJ, which again achieves the highest ever reported energy efficiency.

The contributions of this paper are summarized as follows:

- Dedicated ASIC hardware accelerator for SM over $\mathrm{Four}\mathbb{Q}$ is proposed, whose data path is optimized for arithmetic on $\mathbb{F}_{p^2}$.
- An automated instruction scheduling flow is proposed, which exploits the Python scripting language and a combinatorial optimization solver to minimize the latency for SM computation.
- The fabricated chip demonstrates that the fastest ever reported SM latency of 10.1 μs and that the highest ever reported energy efficiency of 0.327 μJ/SM.

The remainder of this paper is organized as follows. In Section II, the quick summary of an elliptic curve based cryptography and $\mathrm{Four}\mathbb{Q}$ is provided. Then, Section III gives the architecture of the proposed ASIC accelerator, followed by the detail of the automated scheduling optimization flow. The chip implementation and measurement results are provided in Section IV. Finally, conclusion remarks are provided in Section V.

II. Preliminaries

A. Digital signature algorithms

DSA is a digital counterpart of handwritten signatures and stamped seals, which gives a proof that the message was created by a known sender and that was not altered in transit. DSA relies on asymmetric cryptosystems so as to ensure that the signature can be generated only by a sender while its verification can be performed by anonymous receiver. Until recently, the best practice to realize the asymmetric cryptosystems is to exploit the practical difficulty of factoring large integers, e.g. RSA. However, due to rapid improvements in computing capabilities and developments of efficient factorization algorithms, the recommended key size for RSA has been continued to increase and hence the switch from RSA to ECDSA is highly recommended [4]. Following summarizes the workflow of ECDSA based signature generation and verification.

Signature generation: Suppose Alice wants to send a message m to Bob. Initially, they must agree on the following parameters: $Curve$, an elliptic curve used during DSA, G, a base point on the curve, and an integer n such that $[n]G = \mathcal{O}$. Here, $[\cdot]G$ is a scalar multiplication (SM) on the elliptic curve, which is defined as the repeated addition of a point along the curve, i.e., $[n]G = G + G + G + \cdots G$, and $\mathcal{O}$ is the identity element. For latter use, the bit length of n is defined to be L_n. Then, Alice selects a private key $d_A \in [1, n-1]$ uniformly at random and computes a public key as $Q_A = [d_A]G$. The signature of message m can be computed as follows.

1) Calculate $e = HASH(m)$, where $HASH(\cdot)$ is a hash function such as SHA256 [11].
2) Choose an arbitrary integer k from $[1, n-1]$.
3) Calculate $(x_1, y_1) = [k]G$.
4) Calculate $r = x_1 \mod n$. if r is zero, then go back to step 2).
5) Calculate $s = k^{-1}(z + rd_A) \mod n$, where z is the L_n leftmost bits of e. If s is zero, then go back to step 2).

Finally, the pair of (r, s) gives the signature for m. Then, Bob can check whether the signature is valid or not as follows.

Signature verification:

1) Confirm that r and s in $[1, n-1]$. Otherwise, the signature is invalid.
2) Calculate $e = HASH(m)$ and $w = s^{-1} \mod n$.
3) Calculate $u_1 = zw \mod n$ and $u_2 = rw \mod n$, where z is the L_n leftmost bits of e.
4) Calculate $(x_1, y_1) = [u_1]G + [u_2]Q_A$. If $(x_1, y_1) = \mathcal{O}$, then the signature is invalid.
5) The signature is valid if $r = x_1 \mod n$. Otherwise, the signature is invalid.

The security of elliptic curve based cryptosystems is based on the practical difficulty of retrieving the scalar k from $Q = [k]G$ given Q and G. This problem is known to be "discrete logarithm problem" whose difficulty can be analogously illustrated by the integer multiplication over a finite ring $\mathbb{F}_p$ as follows. Given an integer i, an accumulator $acc \in \mathbb{F}_p$ being initialized to one, and a prime number p, let us repetitively multiply i to acc. Suppose that the multiplication is repeated for k times and that k is large enough, it is quite difficult to estimate k only from acc, i, and p since the modulo operation reduces acc into $[0, p-1]$ each time when acc becomes greater than p and one may not know the number of reductions performed only from the available information.

The general and fast algorithm to compute the SM is a repetitive double-and-add method, i.e., to compute $[k]P$, we first start with the binary representation of k: $k = k_0 + 2k_1 + 2^2k_2 + \cdots + 2^{K-1}k_{K-1}$, where $k_i \in \{0, 1\}$ and K is the bit length of k. Then, with the two temporal points N and Q being initialized to P and $\mathcal{O}$, the point addition and doubling are repetitively applied: for $i = 1, 2, \cdots, K-1$, the point addition ($Q \leftarrow Q + N$) is performed when $d_i = 1$, followed by the point doubling ($N \leftarrow N + N$).

Although the ECDSA comes with many advantages, such as the reduced key sizes, its computational burden is relatively high, e.g., a study shows that typical ECDSA implementations spend from 94% to 99% of the time in SM [12], and hence intensive research have been conducted to develop a high-performance elliptic curve on which point arithmetic can be efficiently computed.

B. Four$\mathbb{Q}$

Four$\mathbb{Q}$ is an elliptic curve having approximately 128 bit security level recently proposed by Costello et al. [7], which is defined as the complete twisted Edwards curve over the quadratic extension field $\mathbb{F}_{p^2}$ with $p = 2^{127} - 1$:

$$\xi/\mathbb{F}_{p^2} : -x^2 + y^2 = 1 + dx^2y^2, \quad (1)$$

where $d = 125317048443780598345676279555970305165 \cdot i + 42058576488057777687 70$ and $i^2 = -1$. The algorithm for computing SM over Four$\mathbb{Q}$ is summarized in Algorithm 1, whose unique features are summarized as follows.

1) Use of the quadratic extension field: Although many conventional elliptic curves are defined over $\mathbb{F}_p$, the points on Four$\mathbb{Q}$ are represented as the quadratic extension field $\mathbb{F}_{p^2}$, i.e., a pair of 127 bit integer, a and b, forms a 254 bit element of $\mathbb{F}_{p^2}$: $x = a + b \cdot i$. The field addition and multiplication over $\mathbb{F}_{p^2}$ are similar to those over the complex numbers, i.e., given two elements in $\mathbb{F}_{p^2}$, $a = a_0 + i \cdot a_1$ and $b = b_0 + i \cdot b_1$, where a_0, a_1, b_0 and b_1 are elements in $\mathbb{F}_p$, $a+b$ and $a \times b$ are defined as: $a + b = (a_0 + b_0) + i \cdot (a_1 + b_1)$ and $a \times b = (a_0 + i \cdot a_1) \times (b_0 + i \cdot b_1) = (a_0b_0 - a_1b_1) + i \cdot (a_0b_1 + a_1b_0) = c_0 + i \cdot c_1$, where c_0 and c_1 are the elements in $\mathbb{F}_p$.

2) Use of Mersenne prime as characteristic number: The advantage of employing the Mersenne prime as the characteristic is that the modular multiplication can be implemented without using any integer division, which is the most expensive operation in hardware. Given two integers a and b, the classical modular multiplication computes $a \times b \mod p$ by computing the double-width product $z = a \times b$, followed by the reduction in modulo p, i.e., multiples of p are subtracted from z until the remainder becomes again less than p. Let us derive more efficient modular multiplication by rewriting z as: $z = u \cdot 2^{127} + v$, where u and v are 127 bit integers. Then, it follows from $2^{127} \equiv 1 \mod p$ that $z \equiv u + v \mod p$, and hence the modular reduction can be performed only by a 127 bit modular addition, which is quite efficient compared to the classical approach.

3) Adoption of scalar decomposition technique: Since Four$\mathbb{Q}$ is a degree-2 $\mathbb{Q}$-curve with complex multiplication, there are two efficiently computable endomorphisms, ϕ and ψ with which a 256 bit scalar k can be decomposed into a vector of four 64 bit scalars: $k \mapsto (a_1, a_2, a_3, a_4)$, where a_1 is an odd number [7]. Hence, the scalar multiplication can be rewritten as:

$$[k]P = [a_1]P + [a_2]\phi(P) + [a_3]\psi(P) + [a_4]\phi(\psi(P)). \quad (2)$$

Note here that $[a_1]P$, $[a_2]\phi(P)$, $[a_3]\psi(P)$, and $[a_4]\phi(\psi(P))$ can be computed independently and that the bit length of $[a_i]$ are quarter the bit length of the original scalar k. Hence, the number of iterations in the double-and-add algorithm can be reduced to 1/4 compared to that required for the original k with the cost of several point additions.

III. Proposed Architecture

A. Overview

Fig. 1(a) shows a block diagram of the proposed ASIC cryptoprocessor which composed of a register file, a controller, an $\mathbb{F}_{p^2}$ multiplier, and an $\mathbb{F}_{p^2}$ adder/subtractor. The register file is equipped with four-read and two-write ports so as to minimize the memory access overhead. Further, the datapath is also equipped with forwarding paths so as to enable arithmetic units to be directly fed by their immediate outputs.

Algorithm 1 Scalar multiplication on Four$\mathbb{Q}$

Require: Point $P \in \xi(\mathcal{F}_{p^2})[\xi]$ and integer scalar $k \in [0, 2^{256} - 1]$.
Ensure: $[k]P$.
1: Compute $\phi(P)$, $\psi(P)$ and $\psi(\phi(P))$.
2: Compute $T[u] = P + [u_0]\phi(P) + [u_1]\psi(P) + [u_2]\psi(\phi(P))$ for $u = (u_2, u_1, u_0)_2$ in $[(0,0,0),(0,0,1),\cdots,(1,1,1)]$. Write $T[u]$ in coordinates $(X+Y, Y-X, 2Z, 2dT)$.
3: Decompose k into the tuple of 64 bit scalars (a_1, a_2, a_3, a_4).
4: Recode (a_1, a_2, a_3, a_4) into $(d_{64}, \cdots, d_0)$ and $(m_{64}, \cdots, m_0)$ using Alg.1 in [7].
5: $s_i \leftarrow 1$ if $m_i = -1$ and $s_i \leftarrow -1$ if $m_i = 0$.
6: $Q = s_{64} \cdot T[v_{64}]$
7: **for** $i = 63$ **downto** 0 **do**
8: $\quad Q = [2]Q$
9: $\quad Q = Q + s_i \cdot T[v_i]$
10: **end for**

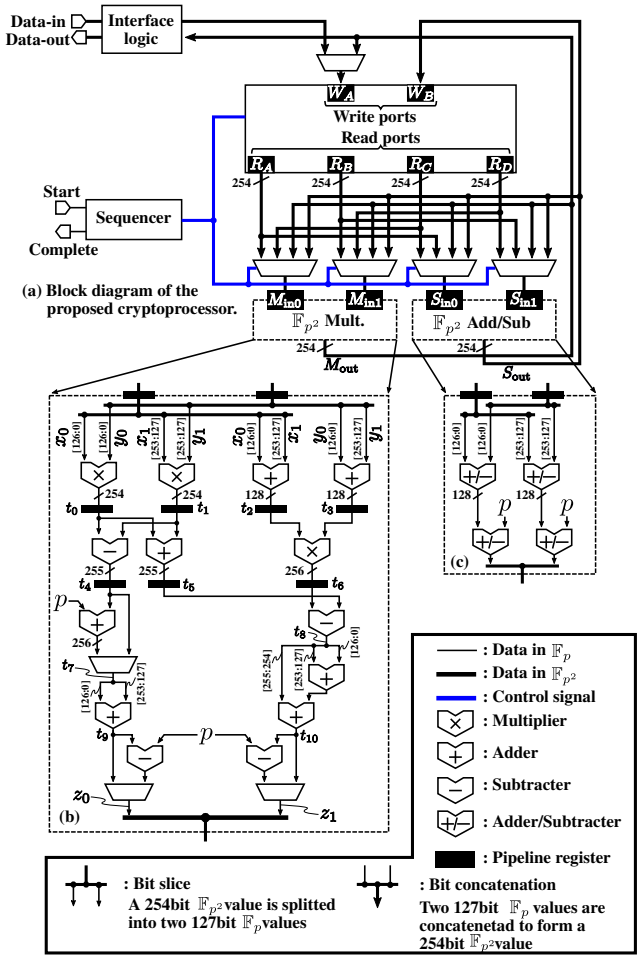

Fig. 1. Processor architecture.

B. Pipelined Karatsuba $\mathbb{F}_{p^2}$ multiplier

Our in-house profiling of Four$\mathbb{Q}$'s SM revealed that $\mathbb{F}_{p^2}$ multiplications accounts for 57% of the total arithmetic operations performed during the SM and hence the acceleration of $\mathbb{F}_{p^2}$ multiplication is crucial for faster SM computation. According to this observation, we choose a pipelined $\mathbb{F}_{p^2}$ multiplier to achieve the throughput of single $\mathbb{F}_{p^2}$ multipli-

Algorithm 2 Karatsuba multiplication over $\mathbb{F}_{p^2}$.

Require: $x = x_0 + x_1 i$ and $y = y_0 + y_1 i$
Ensure: $z = x \cdot y = c_0 + c_1 i$
1: $t_0 \leftarrow x_0 \times y_0$
$t_1 \leftarrow x_1 \times y_1$
$t_2 \leftarrow x_0 + x_1$
$t_3 \leftarrow y_0 + y_1$
2: $t_4 \leftarrow t_0 - t_1$
$t_5 \leftarrow t_0 + t_1$
$t_6 \leftarrow t_2 \times t_3$
3: $t_7 \leftarrow t_4 + p$ **if** $t_4 < 0$ **else** t_4
$t_8 \leftarrow t_6 - t_5$
$t_9 \leftarrow t_7[126:0] + t_7[253:127]$
$t_{10} \leftarrow t_8[126:0] + t_8[253:127] + t_8[255:254]$
$z_0 \leftarrow t_9$ **if** $t_9 < p$ **else** $t_9 - p$
$z_1 \leftarrow t_{10}$ **if** $t_{10} < p$ **else** $t_{10} - p$

cation per a clock cycle. The algorithm for computing $\mathbb{F}_{p^2}$ multiplication is shown in Algorithm 2, whose circuit implementation is shown in Fig. 1(b). To improve the efficiency, we employed Karatsuba multiplication combined with the lazy reduction technique [13], [14].

The lazy reduction is first introduced in the pairing computation by Scott et al. in [13] and later generalized by Aranha et al. in [14]. Studying the multiplication of $\mathbb{F}_{p^2}$ described in Sec. II-B, we notice that computations of pattern like $C = \sum A_i B_i \mod p$ are frequently required. A naive method for calculating C is to conduct the reduction followed by summation, i.e. $C = \sum((A_i B_i) \mod p)$. By applying the lazy reduction technique, the reduction can be "delayed" to the end of the summation, i.e. $C = \sum(A_i B_i) \mod p$, which is known to be quite efficient both in hardware and in software implementations.

Along with the lazy reduction, we adopt Karatsuba multiplication to reduce the number of $\mathbb{F}_p$ multiplication. In the traditional $\mathbb{F}_{p^2}$ multiplier, such as one used in [15], four $\mathbb{F}_p$ multiplications are required to compute a single $\mathbb{F}_{p^2}$ multiplication, i.e., given two $\mathbb{F}_{p^2}$ elements ($A = a_0 + i \cdot a_1$ and $B = b_0 + i \cdot b_1$), $A \cdot B$ can be calculated as $A \cdot B = a_0 b_0 - a_1 b_1 + i \cdot (a_0 b_1 + a_1 b_0)$. On the other hand, Karatsuba multiplication requires only three $\mathbb{F}_p$ multiplications at the cost of a few extra additions: i.e., given $c_0 = a_0 b_0$ and $c_1 = a_1 b_1$ being computed with two $\mathbb{F}_p$ multiplications, $a_0 b_1 + a_1 b_0$ can be calculated with only a single $\mathbb{F}_p$ multiplication and four $\mathbb{F}_p$ addition/subtractions since $a_0 b_1 + a_1 b_0 = (a_0 + a_1)(b_0 + b_1) - c_0 - c_1$.

C. Instruction scheduling

The instruction sequencer is composed of a program ROM that stores the control signals for the datapath and a finite state machine (FSM). Since thousands of microinstructions should be issued during FourQ's SM, optimizing instruction schedules are definitely important to fully exploit the available instruction-level parallelism. This procedure is called as *instruction scheduling*. In many conventional studies such as in [10], the instruction schedules have been designed manually, which is considered to be error-prone. Further, in order to solve the instruction scheduling problem by hand, the entire sequence of the thousands of microinstructions should be divided into multiple small blocks having only tens of microinstructions to simplify the problem, which results in the local optima due to the reduced scheduling flexibility.

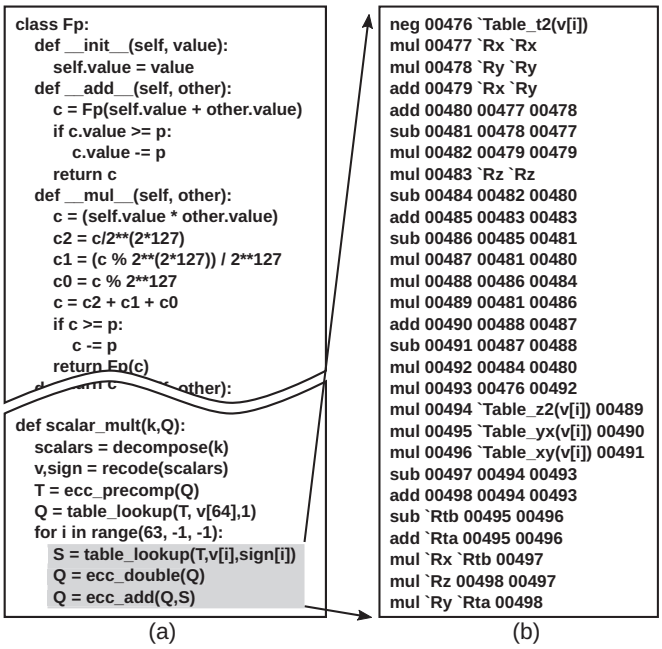

Fig. 2. (a) Code snippet of Python-based pairing implementation and (b) an example microinstruction sequence for $\mathbb{F}_{p^2}$ multiplication.

In order to solve those limitations originated from the hand scheduling optimization, we developed a fully automated design flow by using Python scripting language and a combinatorial optimization solver, whose procedure is summarized as follows.

Step 1) Target algorithm implementation: First, the algorithm for computing SM on FourQ is written by using Python scripting language, which enable us to exploit the high-level abstractions provided by Python such as "if-then-else" or "for" statements. An example code snippet of the Python-based SM implementation is shown in Fig. 2(a).

Step 2) Conversion from high-level instructions into sequences of microinstructions: By recording the subroutine calls occurred during the execution of the Python-based SM algorithm, we can obtain sequences of microinstructions executed during FourQ's SM. Fig. 2(b) shows an example sequence of microinstructions executed during the double-and-add loop, which composed of 15 $\mathbb{F}_{p^2}$ multiplications and 13 $\mathbb{F}_{p^2}$ addition/subtractions.

Step 3) Scheduling optimization: The instruction scheduling can be considered as a job-shop scheduling problem, one of the best known combinatorial optimization problem. Given n tasks and m machines, the job-shop scheduling solver finds an assignments of the tasks to the machines trying to minimize the total processing time. In the developed flow, the dependencies among $\mathbb{F}_{p^2}$ microinstructions are extracted from the traces obtained in **Step 2**, which are converted into the job-shop optimization problems by using PySchedule [16] and finally solved by CP optimizer, a commercial combinatorial optimization solver developed by IBM.

Step 4) Control signal generation: According to the scheduled results, control signals for the datapath is automatically generated. Tab. I shows the scheduled result that corresponds to the code snippet shown in Fig. 2(b).

TABLE I
EXAMPLE INSTRUCTION SCHEDULING RESULT OF THE DOUBLE-AND-ADD LOOP.

Cycle	Register file				$\mathbb{F}_{p^2}$ Mult.			$\mathbb{F}_{p^2}$ Add/Sub				Write back	
	R_A	R_B	R_C	R_D	M_{in0}	M_{in1}	M_{out}	S_{in0}	S_{in1}	cmd.	S_{out}	W_A	W_B
1		$T_{2dT}[v_i]$	Q_z	Q_z									
2	Q_x	Q_y	Q_y	Q_y	Q_z	Q_z			$T_{2dT}[v_i]$	$-$	$-T_{2dT}[v_i]$		
3			Q_x	Q_x	Q_y	Q_y		Q_x	Q_y	$+$	Q_x+Q_y	$t_0 \leftarrow S_{\text{out}}$	
4					Q_x	Q_x	Q_xQ_z					$t_1 \leftarrow S_{\text{out}}$	
5					t_1	t_1	Q_yQ_y	M_{out}	M_{out}	$+$	$2M_{\text{out}}$		$t_2 \leftarrow M_{\text{out}}$
6							Q_xQ_x					$t_3 \leftarrow S_{\text{out}}$	$t_4 \leftarrow M_{\text{out}}$
7	t_4						t_1t_1	M_{out}	t_4	$+$	$M_{\text{out}}+t_4$		$t_5 \leftarrow M_{\text{out}}$
8								t_4	t_5	$-$	t_4-t_5	$t_6 \leftarrow S_{\text{out}}$	$t_7 \leftarrow M_{\text{out}}$
9	t_3			t_6	S_{out}	t_6		t_7	t_6	$-$	t_7-t_6	$t_8 \leftarrow S_{\text{out}}$	
10			t_8		S_{out}	t_6		t_3	t_8	$-$	t_3-t_8	$t_9 \leftarrow S_{\text{out}}$	
11				t_9	t_8	S_{out}	$S_{\text{out}}\cdot t_6$					$t_{10} \leftarrow S_{\text{out}}$	
12			t_0		t_{10}	t_9	$S_{\text{out}}\cdot t_6$						$t_{11} \leftarrow M_{\text{out}}$
13			$T_{2Z}[v_i]$		t_0	S_{out}	$t_8\cdot S_{\text{out}}$						$t_0 \leftarrow M_{\text{out}}$
14	t_{11}				$T_{2Z}[v_i]$	S_{out}	$t_{10}\cdot t_9$						$t_0 \leftarrow M_{\text{out}}$
15		t_{11}	$T_{X+Y}[v_i]$				$t_0\cdot S_{\text{out}}$	t_{11}	M_{out}	$-$	$t_{11}-M_{\text{out}}$		$t_0 \leftarrow M_{\text{out}}$
16			$T_{Y-X}[v_i]$		t_{11}	S_{out}	$T_{2Z}[v_i]\cdot S_{\text{out}}$	t_0	t_{11}	$+$	t_0+t_{11}	$t_1 \leftarrow S_{\text{out}}$	$t_0 \leftarrow M_{\text{out}}$
17		t_0			$T_{Y-X}[v_i]$	S_{out}		M_{out}	t_0	$-$	$M_{\text{out}}-t_0$	$t_2 \leftarrow S_{\text{out}}$	$t_3 \leftarrow M_{\text{out}}$
18							$t_{11}\cdot S_{\text{out}}$	t_3	t_0	$+$	t_3+t_0	$t_0 \leftarrow S_{\text{out}}$	
19					S_{out}	t_0	$T_{Y-X}[v_i]\cdot S_{\text{out}}$					$t_1 \leftarrow S_{\text{out}}$	$t_2 \leftarrow M_{\text{out}}$
20		t_2		t_0				M_{out}	t_2	$-$	$M_{\text{out}}-t_2$		$t_3 \leftarrow M_{\text{out}}$
21				t_1	S_{out}	t_0	$S_{\text{out}}\cdot t_0$	t_3	t_2	$+$	t_3+t_2	$Q_{T_b} \leftarrow S_{\text{out}}$	
22					S_{out}	t_1						$Q_{T_a} \leftarrow S_{\text{out}}$	$Q_Z \leftarrow M_{\text{out}}$
23							$S_{\text{out}}\cdot t_0$						
24							$S_{\text{out}}\cdot t_1$						$Q_X \leftarrow M_{\text{out}}$
25													$Q_Y \leftarrow M_{\text{out}}$

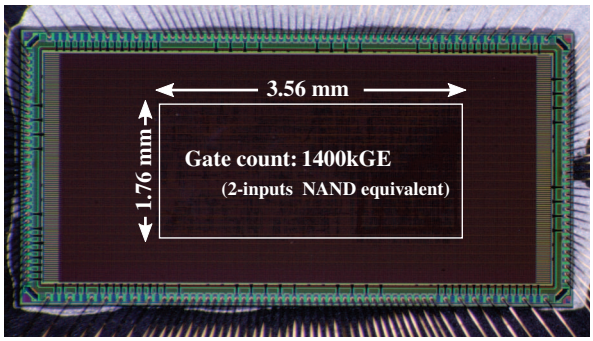

Fig. 3. Chip microphotograph. Our cryptoprocessor occupies 1.76 mm × 3.56 mm silicon area and the circuit complexity is approximately 1400kGE in 2-inputs NAND equivalent.

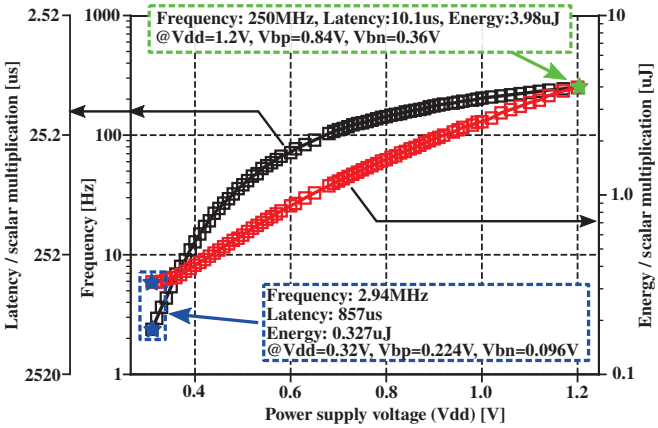

Fig. 4. Measurement result.

IV. EXPERIMENT

A. Circuit implementation

The proposed ASIC crypto processor has implemented using a 65 nm SOTB CMOS process. Fig. 3 shows the fabricated chip microphotograph of a 3.1 mm × 6.1 mm die. The SM unit occupies 1.76 mm × 3.56 mm, which corresponds to 1400 kGE.

B. Measurement result

Fig. 4 summarizes the measurement results of the fabricated processor, where the X and Y axes correspond to the supply voltage and the maximum operating frequency, the latency, and the energy consumption required for computing an SM, respectively. Note that during the measurement, the body bias for pMOS (V_{BP}) and nMOS (V_{BN}) transistors are set to be $0.7\times V_{\text{DD}}$ and $0.3\times V_{\text{DD}}$, where V_{DD} is the supply voltage.

We confirmed that the fabricated chip successfully operated under wide supply voltage range. By applying the typical operating voltage of 1.2 V, our processor successfully computed an SM in 10.1 μs with the energy consumption of 3.98 μJ, at which the fastest SM throughput ever reported has been achieved. Further, by lowering the supply voltage down to 0.32 V, our processor perform an SM with the energy consumption of 0.327 μJ, which again achieves the highest ever reported energy efficiency.

The measurement results with comparisons to the prior art is summarized in Tab. II, demonstrating that the proposed ASIC crypto processor achieved 15.5× and 3.66× faster latency compared to the FourQ implemented on an FPGA platform [10] and P-256 implemented on an ASIC platform [5], respectively. Further, our processor achieved 5.14× higher energy efficiency compared to the conventional ECDSA implemented on an ASIC platform [17].

V. CONCLUSION

The ASIC cryptoprocessor for accelerating SM over FourQ was proposed. By exploiting Karatsuba multiplication and lazy reduction techniques, the datapath of our processor was tailored for $\mathbb{F}_{p^2}$ operations, which was suitable for FourQ's SM. Further, the automated instruction scheduling flow was proposed to fully exploit the available instruction-level parallelism. With the proposed processor fabricated by using a 65 nm SOTB CMOS process, we demonstrated that the fastest ever reported SM latency of 10.1 μs and the highest ever reported energy efficiency of 0.327 μJ/SM were achieved.

ACKNOWLEDGMENT

This paper is based on results obtained from a project commissioned by the New Energy and Industrial Technology Development

TABLE II
COMPARISON TO THE PRIOR ART.

Design	Platform	Curve	# of Cores	Area (A)	V_{DD} [V]	Latency [ms] ♣ (B)	Throughput ♣ [operations / second]	Energy / op. ♣ [μJ]	Latency area product (A)×(B)
Ours	**ASIC 65nm SOTB**	**Four$\mathbb{Q}$**	**1**	**1400kGE**	**0.320**	**0.857**	**117**	**0.327**	**1200**
					1.200	**0.0101**	$\mathbf{9.90 \times 10^4}$	**3.98**	**14.1**
[5] ♠[a]	NANGATE 45nm	NIST P-256	1	1030kGE	—	0.0370	2.70×10^4	—	38.1
				373kGE		0.0750	1.33×10^4		28.0
				322kGE		0.0760	1.32×10^4		24.5
				253kGE		0.115	8700		29.1
				223kGE		0.212	4720		47.3
[18]◇[b]	ASIC 65nm SOTB	Any	1	2490kGE	—	0.0600	1.67×10^4	10.7	149
[17]◇	ASIC 65nm SOTB	Any	1	1.92 mm^2	1.10	0.325	3080	13.9	—
					0.300	2.30	435	1.68	—
[19]◇	Virtex-4	NIST P-256	1	1715 LS 32 DSPs	—	0.495	2020	—	—
			16	24574 LS 512 DSPs		N/A	2.47×10^4		
[20]♦	Virtex-5	NIST P-256	1	1980 LS 7 DSPs 2 BRAMs	—	3.95	253	—	—
[21]♦	Virtex-5	NIST P-256	1	4505 LS 16 DSPs	—	0.570	1750	—	—
[22]♦	Zynq-7020	Curve25519	1	1029 LS 20 DSPs	—	0.397	2520	—	—
			11	11277 LS 220 DSPs		0.341	3.23×10^4		
[10]♦	Zynq-7020	Four$\mathbb{Q}$	1	1691 LS 27 DSPs 10 BRAMs	—	0.157	6390	—	—
			11	5967 LSs 187 DSPs 110 BRAMs		0.170	6.47×10^4		

♣ Latency, throughput, and energy are measured for performing a single operation (signature generation/verification, or scalar multiplication).
♠ The performance of the signature verification is reported.
◇ The performance of the signature generation is reported.
♦ The performance of the scalar multiplication is reported.
[a] The performance is estimated from the post-synthesis simulation.
[b] The performance is estimated from the post-layout simulation.

Organization (NEDO) and is partially supported by JSPS KAKENHI Grant No. 18K13800. This study was conducted through a dedicated licensing program provided by the VLSI Design and Education Center (VDEC) at the University of Tokyo with the cooperation of Cadence Design Systems, Inc., Synopsys, Inc., and Mentor, a Siemens business. The VLSI chip used in this study has been fabricated through VDEC in collaboration with Renesas Electronics Corp.

REFERENCES

[1] M. Barth and K. Boriboonsomsin, "Real-World CO_2 Impacts of Traffic Congestion," *Transportation Research Record*, vol. 2058, no. 1, pp. 163–171, 2008.
[2] "ETSI TS 103 097 V1.3.1 (2017-10) — Intelligent Transport Systems (ITS); Security; Security header and certificate formats," https://www.etsi.org/deliver/etsi_ts/103000_103099/103097/01.03.01_60/ts_103097v010301p.pdf, Accessed: 2018-09-07.
[3] R. L. Rivest, A. Shamir, and L. Adleman, "A Method for Obtaining Digital Signatures and Public-key Cryptosystems," *Commun. ACM*, vol. 21, no. 2, pp. 120–126, Feb. 1978.
[4] NIST, "Recommendation for Key Management, Part 1: General," Jan. 2016.
[5] M. Knežević, V. Nikov, and P. Rombouts, "Low-Latency ECDSA Signature Verification — A Road Toward Safer Traffic," *IEEE Trans. VLSI Syst.*, vol. 24, no. 11, pp. 3257–3267, Nov 2016.
[6] "IEEE 5G and Beyound Technology Roadmap White Paper — October 2017," https://futurenetworks.ieee.org/images/files/pdf/ieee-5g-roadmap-white-paper.pdf, Accessed: 2018-09-13.
[7] C. Costello and P. Longa, "Four$\mathbb{Q}$: Four-Dimensional Decompositions on a $\mathbb{Q}$-curve over the Mersenne Prime," in *Int. Conf. on Advances in Cryptology — ASIACRYPT*, 2015, pp. 214–235.
[8] R. P. Gallant, R. J. Lambert, and S. A. Vanstone, "Faster Point Multiplication on Elliptic Curves with Efficient Endomorphisms," in *Int. Conf. on Advances in Cryptology — CRYPTO*, 2001, pp. 190–200.
[9] D. J. Bernstein, "Curve25519: New Diffie-Hellman Speed Records," in *Int. Workshop on Public Key Cryptography*, 2006, pp. 207–228.
[10] K. Järvinen, A. Miele, R. Azarderakhsh, and P. Longa, "Four$\mathbb{Q}$ on FPGA: New Hardware Speed Records for Elliptic Curve Cryptography over Large Prime Characteristic Fields," in *Cryptographic Hardware and Embedded Syst.*, Aug 2016, pp. 517–537.
[11] "Federal Information Processing Standard (FIPS) 180-4 — Secure Hash Standard (SHS)," https://ws680.nist.gov/publication/get_pdf.cfm?pub_id=910977, Accessed: 2018-09-07.
[12] A. M. Fiskiran and R. B. Lee, "Evaluating instruction set extensions for fast arithmetic on binary finite fields," in *Int. Conf. on Application-Specific Syst., Architectures and Processors*, Sept 2004, pp. 125–136.
[13] M. Scott, "Implementing Cryptographic Pairings," in *Int. Conf. on Pairing-Based Cryptography*, 2007, pp. 177–196.
[14] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, "Faster Explicit Formulas for Computing Pairings over Ordinary Curves," in *Int. Conf. on the Theory and Appl. of Cryptographic Techn.*, 2011, pp. 48–68.
[15] J. Han, Y. Li, Z. Yu, and X. Zeng, "A 65 nm Cryptographic Processor for High Speed Pairing Computation," *IEEE J. Solid-State Circuits*, vol. 23, no. 4, pp. 692–701, April 2015.
[16] "PySchedule," https://github.com/timnon/pyschedule.
[17] M. Tamura and M. Ikeda, "1.68μj/Signature-Generation 256-bit ECDSA over GF(p) Signature Generator for IoT Devices," in *Asian Solid-State Circuits Conf.*, Nov 2016, pp. 341–344.
[18] M. Tamura and M. Ikeda, "Montgomery Multiplier Design for ECDSA Signature Generation Processor," *IEICE Trans. on Fundamentals of Electron., Commun. and Comput. Sci.*, vol. E99.A, no. 12, pp. 2444–2452, 2016.
[19] T. Güneysu and C. Paar, "Ultra High Performance ECC over NIST Primes on Commercial FPGAs," in *Cryptographic Hardware and Embedded Syst.*, 2008, pp. 62–78.
[20] K. C. C. Loi and S. Ko, "Scalable Elliptic Curve Cryptosystem FPGA Processor for NIST Prime Curves," *IEEE Trans. VLSI Syst.*, vol. 23, no. 11, pp. 2753–2756, Nov 2015.
[21] D. B. Roy, D. Mukhopadhyay, M. Izumi, and J. Takahashi, "Tile before multiplication: An efficient strategy to optimize DSP multiplier for accelerating prime field ECC for NIST curves," in *Design Automation Conf.*, June 2014, pp. 1–6.
[22] P. Sasdrich and T. Güneysu, "Implementing Curve25519 for Side-Channel–Protected Elliptic Curve Cryptography," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 9, no. 1, pp. 3:1–3:15, Nov. 2015.