

Low-Cost Hardware Implementation of Elliptic Curve Cryptography for General Prime Fields

Yuan Ma^{1,2}(✉), Qinglong Zhang³, Zongbin Liu^{1,2},
Chenyang Tu^{1,2}, and Jingqiang Lin^{1,2}

¹ Data Assurance and Communication Security Research Center,
Chinese Academy of Sciences, Beijing, China
{yma,zbliu,chytu,linjq}@is.ac.cn

² State Key Laboratory of Information Security,
Institute of Information Engineering,
Chinese Academy of Sciences, Beijing, China

³ Huawei Technologies, Zhejiang, China
ql.zhang@huawei.com

Abstract. In resource-constrained applications, elliptic curve cryptography (ECC) is preferable for the property of shorter key size with comparable security. Binary extension fields are usually used for area-optimized implementations, since the complex carry-propagation logics are avoided over these fields. However, efficient ECC implementations over (general) prime fields are still challenging for low-area constraint. As a popular implementation platform for cryptographic algorithms, Field Programmable Gate Array (FPGA) attracts more and more attentions for these applications due to its nice properties of flexibility and short development cycle. In this paper, we propose a compact and efficient arithmetic logical unit (ALU) by highly integrating the functions of Montgomery modular multiplications, additions and subtractions over general prime fields. Then we design a low-cost hardware architecture for generic elliptic curve point multiplications for FPGA platforms. Experimental results indicate that the implementation only occupies 105 Slices, 2 DSP blocks and 2 BRAMs in Spartan-6 FPGA. To the best of our knowledge, our implementation is the smallest for general prime fields in FPGAs.

Keywords: Elliptic curve cryptography · Low-cost · FPGA · Implementation

1 Introduction

Low-cost cryptographic implementations have been more and more attractive for modern applications. A low-cost implementation also means the reduction of

Q. Zhang—This work was performed while the second author was in Chinese Academy of Sciences.

consumed resources and power, which is necessary for these constrained scenarios. Although the available resources are strictly constrained, public-key cryptography (PKC) involving burdensome arithmetic is usually required due to its advantages over symmetric cryptography. Compared with RSA or other PKC algorithms over finite fields, elliptic curve cryptography (ECC) uses a much shorter key to achieve an equivalent level of security. Therefore, ECC implementations are preferred for resource-constrained applications owing to the lower computational complexity and other nice properties such as reduced storage and power consumption. Compared with software and ASIC (Application Specific Integrated Circuit) implementations of ECC, FPGA implementation is a better choice in the trade-off between execution speed and development period. In addition, FPGAs are often used as the early-validation platforms for the ASICs, thus have important research values.

ECC designs can be roughly split into two categories: over (extended) binary fields \mathbb{F}_{2^n} and over prime fields \mathbb{F}_p , and also some designs support both the two. The first category offers better performance and lower resource consumption, mostly because no carry is propagated in the field structure. That is why most area-optimized designs focus on binary field implementations. Nevertheless, prime fields also have significant value either in the applications, such as digital signature generation, or in the standards of elliptic curves [4, 5, 16]. The NIST curves over binary or prime fields have significant advantages in area and speed for hardware implementations, as fast reduction methods can be adopted and the parameters are fixed. Therefore, most of the low-area targeted designs [1, 6, 8, 11, 18, 19, 21, 23] (especially in ASICs) are focusing on these standardized curves. These implementations are efficient for elliptic curve point multiplications (ECPMs), but for further applications (such as digital signature generation), they are hard to be complete these functions alone. For example, the Elliptic Curve Digital Signature Algorithm (ECDSA) needs another prime (the order of the base point) field operations for the final operation, but fast reduction methods cannot be applicable for that prime which is not special. Existing implementations based on the fast reduction have to specifically add the scheduling instructions and improve the hardware arithmetic unit for supporting the signature generation, such as [18].

In this paper, in order to better support the applications based on ECC (such as curve transition, or signature generation and key agreement which require the operations over another prime field) for area-constrained scenarios, we propose a low-cost hardware ECC implementation for general prime fields based on Montgomery modular multiplications. Besides supporting the operation over another prime field under the same curve, the implementation also supports the transition of different curves (of the same length) without reconfiguration. For example, if needed, the user can switch the underlying curve from the NIST curve P-256 to the SM2 elliptic curve (Chinese ECC standard [16]) through writing new parameters for meeting certain demands.

The implementation is constructed based on our proposed prime-field arithmetic logical unit (ALU) for modular arithmetic, which is able to perform the operations of Montgomery modular multiplications, additions and subtractions

with high compatibility. For the sake of efficiency, the ALU is designed to be a high-radix architecture. Particularly, we employ Shift Register Look-up Table (SRLs) in FPGAs to implement long registers to minimize the occupied area, and eliminate the operations between additions/subtractions and Montgomery multiplications to decrease the control circuits. Furthermore, we maximize the frequency of the ALU with the help of the dedicated DSP (Digital Signal Processing) blocks in modern FPGAs, and the execution efficiency of the ECC implementation is significantly improved. In the higher level, we optimize the scheduling process in the point addition to improve the use efficiency of the ALU. Finally, we implement the design in Spartan-6 FPGA platform. The design only occupies 105 Slices, 2 DSP blocks and 2 BRAMs in Spartan-6 FPGA with a low computation latency. Comparison results indicate that our implementation outperforms the existing works in FPGAs over prime fields in the aspect of area, and a lot of logic Slices or dedicated cores are saved.

The rest of this paper is organized as follows. Section 2 presents the preliminaries for elliptic curve cryptography. Section 3 describes the hardware architectures of the ALU and the ECC processor. Section 4 gives implementation results in FPGAs. Section 5 presents the comparison results with related work. Section 6 concludes the paper.

2 Elliptic Curve Cryptography

The elliptic curve is defined over a field \mathbb{K} given by the Weierstrass equation:

$$E : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6. \quad (1)$$

In the case of the characteristic $Char(\mathbb{K}) \neq 2, 3$, the general Weierstrass equation is simplified to

$$E : y^2 = x^3 + ax + b. \quad (2)$$

In [14], Montgomery developed an original technique to compute multiples of points on an elliptic curve, as shown in Algorithm 1. His technique is based on the fact that the sum of two points whose difference is a known point can be computed without the y -coordinates of the two points. For each iteration in the Montgomery ladder algorithm, $Q_1 - Q_0$ always equals to the base point P , thus can be preset as a constant in advance. As the speed is not the primary optimization goal, we do not adopt more efficient algorithms such as NAF (Non-adjacent Form) or window-based algorithms, but employ the Montgomery ladder algorithm whose control is simpler. In addition, the method is resistant against Simple Power Analysis (SPA).

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2) \in E(\mathbb{F}_q)$ with $P \neq \pm Q$, and $P + Q = (x_3, y_3)$, $2P = (x_4, y_4)$. Given the point $P - Q = (x', y')$, the x -coordinates of $P + Q$ and $2P$ satisfy [2]:

$$x_3 = \frac{2(x_1 + x_2)(x_1x_2 + a) + 4b}{(x_1 - x_2)^2} - x', \quad (3)$$

Algorithm 1. Montgomery ladder algorithm for point multiplication

Input: $P \in E(\mathbb{F}_q)$, $k = (k_{l-1}, \dots, k_1, k_0)$ with $k_{l-1} = 1$
Output: kP
1: $Q_0 = P; Q_1 = 2P$
2: **for** $i = l - 2$ **down to** 0 **do**
3: **if** $k_i = 0$ **then**
4: $Q_1 = Q_0 + Q_1; Q_0 = 2Q_0$
5: **else**
6: $Q_0 = Q_0 + Q_1; Q_1 = 2Q_1$
7: **end if**
8: **end for**
9: **return** (Q_0)

and

$$x_4 = \frac{(x_1^2 - a)^2 - 8bx_1}{4(x_1^3 + ax_1 + b)}. \tag{4}$$

The formulas for point addition and point doubling require a field inversion and several field multiplications. If inversion in \mathbb{K} is significantly more expensive than multiplication, then it may be advantageous to represent points using projective coordinates.

The projective point $(X : Y : Z), Z \neq 0$, corresponds to the affine point $(X/Z, Y/Z)$. The projective equation of the elliptic curve is

$$Y^2Z = X^3 + aXZ^2 + bZ^3.$$

The point at infinity ∞ corresponds to $(0 : 1 : 0)$, while the negative of $(X : Y : Z)$ is $(X : -Y : Z)$. Under the standard projective coordinate, these equations becomes [10]:

$$Z_3 = (X_1Z_2 + X_2Z_1)^2, \tag{5}$$

$$X_3 = 2(X_1Z_2 + X_2Z_1)(X_1X_2 + aZ_1Z_2) + 4bZ_1^2Z_2^2 - x'Z_3, \tag{6}$$

$$Z_4 = 4Z_1(X_1^3 + aX_1Z_1^2 + bZ_1^3), \tag{7}$$

$$X_4 = (X_1^2 + aZ_1^2)^2 - 8bX_1Z_1^3. \tag{8}$$

In some cases (such as public key generation), the y -coordinate is required. The y -coordinate of P can be deduced by:

$$y_1 = \frac{2b + (a + x'x_1)(x' + x_1) - x_2(x' - x_1)^2}{2y'}.$$

Note that y' is the y -coordinate of the base point that is known in advance in Algorithm 1, so $(2y')^{-1}$ can be pre-computed and the inversion is replaced by a multiplication in the equation.

In the original ladder, the addition and doubling are computed separately. For sake of efficiency, Izu et al. [9] encapsulated these formulae into one formula, which outputs x -coordinate values of $P + Q$ and $2P$ on inputs P and Q . With a projective version of the x -coordinate-only formulae, X_3, Z_3, X_4, Z_4 can be computed with 17 multiplications and 18 additions. The number of auxiliary variables for the formulae is 7. The y -coordinate recovery algorithm requires 13 multiplications and 7 additions and 7 auxiliary variables [9]. For one ECPM, the y -coordinate recovery is only performed once, thus has negligible impact on the execution efficiency.

3 Hardware Architecture

In general, the compact ECC architecture is usually composed of four parts: the ALU for finite-field arithmetic, data memory, program memory and other control circuits. The core of the design is to simplify the ALU operations and further utilize the ALU efficiently for elliptic curves arithmetic. In this section, we first propose a compact and efficient ALU architecture based on a series of design policies, and then design a low-cost ECC architecture.

3.1 ALU Design

Design Policy. The design of an ALU is crucial for ECC implementations. The purpose of our design is to construct a compact and area-saving architecture. To achieve this goal, we establish the following design policy for the circuit architecture.

1. To support the operations of general prime fields or generic curves rather than specific curves (such as NIST curves) in order to guarantee the flexibility.
2. To set the bus width of the input/output signals to no more than 16 bits in order to simplify the control, and to use single-port RAM rather than dual-port RAM.
3. To integrate all the prime-field operations to use the same computing circuit in order to save consumed resources.
4. To specifically enrich the ALU functions to optimize the scheduling process inside the elliptic curve arithmetic.

Item (1) ensures the flexibility of the ECC implementation, so that it is not only suitable for the NIST primes (such as P-192, P-256) but also for the operations over general prime fields. Item (2) guarantees lower-resource consumption when using RAMs. With narrower width and single port RAM, smaller area of control circuits are consumed. The use of single-port RAM is to improve the transportability of the architecture. Item (3) makes the same circuit perform modular multiplication, addition and subtraction by configuration, which improves the utilization. Item (4) enhances the efficiency for calculating point addition and doubling, as some involved specific operations can be optimally executed by the ALU.

Modular Arithmetic. According to the design policy (1), we choose Montgomery multiplication as the underlying modular multiplication algorithm, rather than the fast reduction method which is only available for pseudo Mersenne primes such as NIST standardized primes [5]. Montgomery multiplication is a method to perform modular multiplication without the need to perform division by the modulus [13]. A version of Montgomery’s algorithm [17] is given as Algorithm 2. This algorithm avoids multiplication and addition in quotient determination to simplify the computation.

Algorithm 2. Montgomery multiplication algorithm with simplified quotient determination [17]

Input:

- A modulus $M > 2$ with $\text{gcd}(M,2) = 1$ and positive integers w, n such that $4\widetilde{M} < 2^{wn}$, where \widetilde{M} is given by $\widetilde{M} = (\overline{M} \bmod 2^w)M$.
- Integer R^{-1} , where $(2^{wn}R^{-1}) \bmod M = 1$
- Integer \widetilde{M} , where $(-M\widetilde{M}) \bmod 2^w = 1$
- Integer multiplicand A , where $0 \leq A \leq 2\widetilde{M}$
- Integer multiplier $B = \sum_{i=0}^n (2^w)^i b_i$, where digit $b_n = 0, b_i \in \{0, 1, \dots, 2^w - 1\}$ for $0 \leq i < n$ and $0 \leq B \leq 2\widetilde{M}$

Output:

- An integer S_{n+1} where $S_{n+1} \equiv ABR^{-1} \pmod{M}$ and $0 \leq S_{n+1} \leq 2\widetilde{M}$
 - 1: $S_0 = 0$;
 - 2: **for** $i = 0$ **to** n **do**
 - 3: $q_i = S_i \bmod 2^w$;
 - 4: $S_{i+1} = S_i \text{ div } 2^w + q_i M' + b_i A$, where $M' = (\widetilde{M} + 1) \text{ div } 2^w$;
 - 5: **end for**
 - 6: return S_{n+1}
-

On the observation from Algorithm 2, the step to calculate S_{i+1} is crucial for this modular multiplication. In order to make this algorithm suitable for hardware implementation, we propose a processing method which uses two w -bit multipliers and a few adders, as shown in Algorithm 3. The long integer S_i and M' are split into w -bit blocks. The remaining inputs appearing in Algorithm 2 are omitted.

In Algorithm 3, S_i and M' are divided into n w -bit blocks. S_i is represented as: $S_i = \{S_i^{(n-1)}, S_i^{(n-2)}, \dots, S_i^{(0)}\}$. During the initialization procedure, S_0 and $(w + 1)$ -bit *Carry* are set to be zero. After the initialization, there are two loops to complete the Montgomery multiplication. For the operands A and B , the outer loop is responsible for the split of B and the inner loop controls the use of w -bit a_j . The most resource consuming calculation is in the inner loop, and this calculation is involved with two w -bit multipliers and an addition with four addends. Since the inner loop consumes n clocks and the outer loop has $(n + 1)$ times, it takes $n(n + 1)$ clocks to finish one Montgomery multiplication.

Algorithm 3. Processing method for the compact design of Montgomery multiplication

Input:
 $A, B, M' = \{m'_{n-1}, \dots, m'_0\}$

Output:
 $S_{n+1} \equiv ABR^{-1} \pmod{M}$ and $0 \leq S_{n+1} \leq 2\widetilde{M}$.

- 1: $S_0 = 0, Carry = \{w + 1\}'b0;$
- 2: **for** $i = 0$ **to** n **do**
- 3: $q_i = S_i^{(0)};$
- 4: **for** $j = 0$ **to** $n - 1$ **do**
- 5: $\{Carry, S_{i+1}^{(j)}\} = S_i^{(j+1)} + Carry + q_i m'_j + b_i a_j;$
- 6: **end for**
- 7: **end for**
- 8: return S_{n+1}

Based on the processing method, we add the modular addition and subtraction function by using the existing accumulator in the multiplier. In fact, the Montgomery multiplication has the function of reduction by expanding R . For example, for any input $A, B < 4\widetilde{M}$, when $R = 2^{wn} > 4 \times 4\widetilde{M} = 16\widetilde{M}$, the final result still satisfies $0 < S_{n+1} < 2\widetilde{M}$. Therefore, we do not perform the reduction of the addition results, but only to guarantee the multiplier input in the range of $(0, 4\widetilde{M})$. For modular subtraction, we need to add integer multiples of M to the subtraction result when it is negative. The algorithm for modular addition and subtraction is shown in Algorithm 4.

Algorithm 4. Modular addition and subtraction sharing the circuit of Montgomery multiplication

Input: A, B, \widetilde{M} ,
operation flag $sub \in \{0, 1\}$ denotes a subtraction when $sub = 1$ and addition otherwise;

Output: S

- 1: $S = 0, Carry = (w + 1)'b0;$
- 2: **for** $j = 0$ **to** $n - 1$ **do**
- 3: $\{Carry, S^{(j)}\} = Carry + b_j \oplus \{sub\} + a_j + (j == 0 ? 1 : 0);$
- 4: **end for**
- 5: **while** $Carry \neq 0$ **do**
- 6: $\{Carry, S\} = \{Carry, S\} + \widetilde{M};$
- 7: **end while**
- 8: return S

In the initialization of Algorithm 4, S_0 and $(w + 1)$ -bit $Carry$ are set to zero. According to the operation flag sub , the accumulator can complete both addition and subtraction. When the operation is modular addition, after the n -time loop,

the operation is completed. When the operation is modular subtraction, the XOR operation is activated and ‘1’ is added in the first round. Furthermore, if *Carry* is not zero after the iteration, which means the final result of S is negative, it requires extra additions with \bar{M} until S is positive.

Design Architecture. The ALU architecture integrating Montgomery multiplication, addition and subtraction operations is depicted as Fig. 1. According to the design policy (2), only one data input port is allowed for the ALU. There are three logic calculation units: two w -bit multipliers and one adder with four inputs. Also, there are two long shift registers which are used for storing M' and the computed result S , and these two registers are shifted w -bit every clock. Here, we carefully minimize the number of states of the shift registers to reduce the control circuits for these massive registers.

The data flow of the multiplication in the ALU is explained as follows. Before the first modular multiplication, we should load the modulus into the shift register M' and the bus width of this data input port is w , so it takes nearly $(n - 1)$ -time shifts to finish the loading. Note that the loading operation is only executed once at the beginning of the prime field computation. Algorithm 3 is performed after rightmost w -bit b_0 into the ALU register. Then a_0, a_1, \dots, a_{n-1} successively enter the left multiplier unit of the ALU figure. The width of the addition result is $2w + 1$ bit, where the higher $w + 1$ bits are fed back to the adder in the next cycle and the rest are put into the S shift registers. Due to the one extra cycle for loading b_i , the total clock cycles for one multiplication increases to $(n + 1)^2$.

The addition and subtraction shares the same accumulator of the ALU. The configuration is easily completed by using the multiplexers which are responsible for switching the values of *sub* and zero. Another advantage of the ALU structure is that it can perform the operation $\alpha A \pm \beta B$, where $\alpha, \beta \in [0, 2^w - 1]$ and $\beta \equiv 1$ for subtraction. The function is more powerful by combining Algorithm 4, as the result can be immediately input to the ALU for Montgomery multiplication without modular reduction gradually. This is useful to merge the adjacent additions and subtractions in point addition and doubling algorithms, thus saves program commands and consumed time. The required clock cycles for one-time addition/subtraction is $2(n + 1)$.

3.2 FPGA Optimization

In modern FPGAs, the dedicated resources and multifunctional logics allows us to further improve the efficiency of the ALU.

Maximizing Frequency. The maximum frequency of the ALU is limited due to the long critical path, which consists of one multiplier, one adder of four numbers and some multiplexers, as shown in Fig. 1. This could reduce the compatibility of the ALU (or the ECC implementation) with high-speed modules that run at a high frequency in the same FPGA. Therefore, we maximize the frequency of

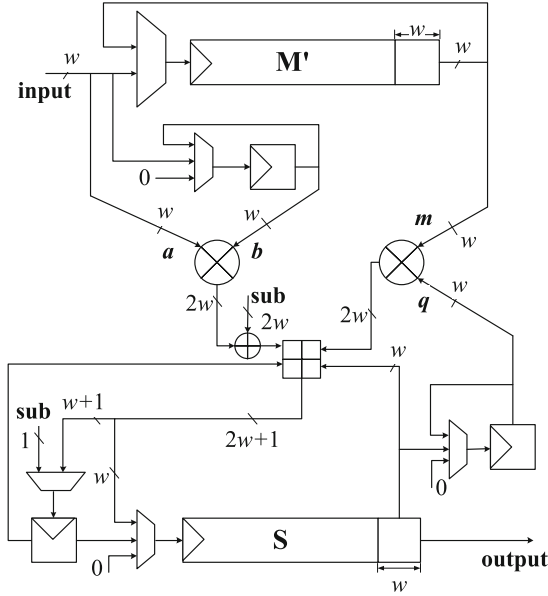


Fig. 1. The ALU architecture

the ALU with the help of the dedicated DSP (Digital Signal Processing) blocks in modern FPGAs, which also significantly improves the execution efficiency of the ECC implementation. In our targeted FPGA platform Xilinx Spartan-6, the DPS block named DSP48A1 is able to perform (mixed) multiplications and subtractions/additions efficiently, and the frequency can achieve very high by inserting the registers (i.e., pipeline) inside DSP blocks.

In the pipeline setting up, to guarantee that the result is computed in each loop after the pipeline is filled, we shall recognize that which variables can be known in advance and which ones cannot be. The critical data path of the ALU in the i -round j -loop is represented as:

$$\{Carry, S_{i+1}^{(j)}\} = Carry + S_i^{(j+1)} + q_i * m'_j + a_i * b_j,$$

where the variables except for $Carry$ for the next loop are known before the end of the current loop, thus can be pre-computed. In addition, the result of $a_i * b_j$ should be negated for the subtraction operation (b_j always equals 1 for the subtraction), and we also put the negation operation into the DSP blocks by being subtracted with w -bit 1's.

The data path with pipelines inside DSP blocks is depicted as Fig. 2, which contains two DSP blocks with a three-stage pipeline. The stage latencies are balanced to maximize the frequency. In the 1st stage, the two multiplications are performed. The negation and the addition operations are processed in the 2nd stage, and the two DSP blocks are connected using the cascade connection ports PCOUT and PCIN to decrease the wire delay. The remain three-number

addition is performed in the 3rd stage. After the frequency optimization, the maximum frequency of ALU is improved to 200 MHz in our experiment, which is nearly three times than the original. The execution time of the multiplication or addition/subtraction slightly increases by two clock cycles that is caused by filling up the pipeline.

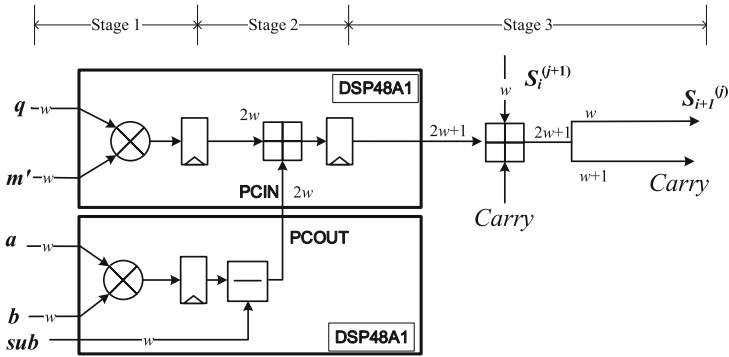


Fig. 2. The optimized data path with pipelines

Minimizing Area. In the ALU, the main consumed resources are divided into three parts: multiplier and adder units, the long shift registers, and the control circuit, and we have put some addition operations in the DSP blocks to improve the resource utilization. Here we further employ the SRLs in FPGAs to implement the long shift registers. SRLs are one type of LUTs that can efficiently implement shift registers. One 6-input SRL can compose 1-bit-width and 64-bit-depth shift registers, while it requires 64 registers in the non-optimizing manner. In addition, as we mentioned, the two groups of shift registers (S and M in Fig. 1) have only two states: suspending and shifting, except for the heading registers that have different inputs. For SRL implementations, each SRL has an input to control the inside registers shifting or not. Therefore, these registers can be efficiently implemented in FPGAs with a ultra-low cost.

3.3 ECC Architecture

Based on the ALU design and the Montgomery ladder algorithm for point multiplication, we design our ECC architecture by utilizing decoders and program commands to control all the required operations over prime fields. The ECC architecture is designed as shown in Fig. 3. Except for the ALU module, two memories are deployed in the architecture. One is the PROGRAM ROM and the other is a DATA RAM which is used for storing intermediate values, the constants, and the final results.

Point multiplication is composed of a series of point addition and point doubling. After the mix of modular multiplications and additions by utilizing the

multiple-and-addition function of the ALU, one point addition and doubling needs 17 modular multiplications and 12 modular addition/subtraction. For each modular multiplication or modular addition/subtraction, there is a command to indicate the type of this operator, the addresses for the operands of this operator. Therefore, for one point addition and point doubling there are 29 commands in total and there is a decoder to determine which command is used for next calculation according to k . In order to simplify the decoder, we have two sets of commands in the PROGRAM ROM, one set of 29 commands is for the case in which k_i is 0, and the other set of 29 commands is for the case in which k_i is 1.

Modular inversion is required for transferring the projective coordinates to affine ones or generating the signature. In our implementation, the operation is carried out by utilizing the ALU, and the commands are also stored in the ROM of Fig. 3. We take advantage of Fermat’s little theorem to calculate the inversion, which is

$$a^{p-2} \equiv a^{-1} \pmod{p}. \tag{9}$$

Hence, the operation of modular inversion just consumes a few extra memory and control circuits.

4 Hardware Implementation

In this section, we implement the 256-bit ECC architecture in Xilinx FPGAs, and evaluate its area and efficiency.

4.1 Efficiency

For demonstration purposes, we implement 256-bit ECPM of generic curves over prime fields in FPGAs. The parameters of the ALU and ECC processor are set as follows. The modulus M has 256-bit length, the width $w = 16$, $n = 18$, and $R = 2^{wn} = 2^{288}$. Hence, the total number of processing cycles for a modular multiplication and addition/subtraction is $(n+1)^2+2 = 363$ and $2(n+1)+2 = 40$,

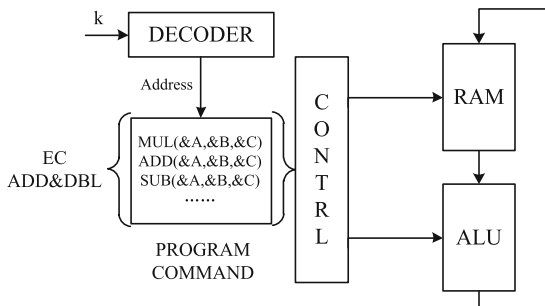


Fig. 3. The ECC architecture

respectively. The RAM size is set to 256-bit depth and 16-bit width (256×16), and the program ROM size is 128×16 .

For 256-bit ECPM, the required clock cycles of $w = 16$ for different operations are listed in Table 1. One 256-bit ECPM has at most 256 point addition & doubling operations for Montgomery ladder algorithm. The inversion in the ECPM is only considered once. The y -coordinate is also calculated at the end of ECPM. As a result, the 256-bit ECPM for generic curves can be completed in about 1847 K clock cycles for $w = 16$.

Table 1. Required clock cycles for the operations ($w = 16$)

Operation	MUL	ADD/SUB	Point ADD & DBL	INV	Other	ECPM
Clock cycles	363	40	6651	139392	5362	1847410

4.2 FPGA Implementation

We implement the ECC processor on Xilinx Spartan-6 (XC6SLX45T-4) and Spartan-3E (XC3S100E-5) FPGAs. The two FPGAs are both low-cost-application oriented, while the latter is an old device which is useful for the fair comparison with previous work. The post place and route (PAR) implementation results by ISE 14.6 are shown in Table 2. Note that the differences between internal structure of the two FPGAs make the consumed logics seemingly significantly different. One Slice in Spartan-3E contains two 4-input LUTs and two flip-flops, while the Slice in Spartan-6 contains four 6-input LUTs and four flip-flops. The multiplier resources are multi-functional DSP units in Spartan-6, while they are multiplier hardcores (MULT) in Spartan-3E. It is noted that the functions that were completed in DSP blocks are implemented using LUT logics in Spartan-3E.

Table 2. PAR results of the ECC processor in FPGAs

Device	Resource					Speed	
	LUT	Flip-flop	Slice	DSP/MULT	BRAM	Frequency	ECPM latency
Spartan-6	332	372	105	2	2	200.4 MHz	9.2 ms
Spartan-3E	520	392	350	2	2	145.6 MHz	12.7 ms

On Spartan-3E FPGA, 350 Slices, 2 MULTs, and 2 BRAMs are occupied to implement our ECC architecture with $w = 16$, including 520 LUTs and 392 Flip-flops. The maximum frequency on Spartan-3E is 145.6 MHz, and it needs 12.7 ms to finish one ECPM. In Spartan-6 FPGA, it costs only 105 Slices (including 332 6-input LUTs and 382 flip-flops), 2 DSP blocks and 2 BRAMs. One ECPM consumes only 9.2 ms with $w = 16$ in Spartan-6. From these results, it is observed that our low-cost ECC architecture has an excellent performance in FPGAs: very small area with a low computation latency.

5 Related Work and Comparison

Many low-cost ECC implementations are targeted to ASICs. Because the area is extremely constrained and the application is specific, the ASIC implementations use standardized curves that support the fast reduction, such as [1, 6, 8, 11, 18, 23]. Different with their underlying methods, we choose the Montgomery multiplication as the modular method, which is more complex than the fast reduction, but is more flexible. This allows us to support generic curves or different prime fields under the same curve without reconfiguration. Hence, we do not perform the comparison with ASIC implementations due to the differences in the underlying platform and the design goal.

Table 3. Comparison of ECC implementations in FPGAs (256-bit prime field)

Work	FPGA	Curve	Slice	MULT/DSP	BRAM	Freq. (MHz)	Latency (ms)
This work (16 bit)	Spartan-6	any	105	2	2	200.4	9.2
	Spartan-3E	any	350	2	2	145.6	12.7
Vliegen [22] (16 bit)	Virtex-2 Pro	any	1832	2	9	108.2	29.83
Varchola [21] (16 bit)	Virtex-2 Pro	P-256	773	1	3	210.0	10.02
Roy [19]	Spartan-6	P-256	72	8	24	156.25	12.2
McIvor [3]	Virtex-2 Pro	any	15755	256	0	39.5	3.84
Tawalbeh [12]	Virtex-5	any	20000	0	0	200	1.66
Ghosh [20]	Virtex-4	any	13661	0	0	43	9.2
Guillermin [15]	Stratix II	any	9177ALM	96	0	157.2	0.68
Ma [24]	Virtex-5	any	1725	37	10	291	0.38
Güneysu [7]	Virtex-4	P-256	1715	32	11	490	0.45

Table 3 lists related works for ECC implementations in FPGAs, where the former three [19, 21, 22] focused on compact implementations and others [3, 7, 12, 15, 20, 24] on high-speed implementations. Especially, [7, 19, 21] optimized the implementation efficiency for NIST prime P-256. Vliegen *et al.* [22] presented a compact FPGA-based architecture for ECC over prime fields by using the coarsely integrated operand scanning (CIOS) method of Montgomery multiplication. The architecture is available for any prime-field curve, but the occupied area is large. Varchola *et al.* [21] optimized the computational unit using fast reduction for the NIST primes, and obtained a very high frequency and relative small area (773 Slices in Virtex-2 Pro). Targeting generic curves, our optimized ECC architecture has a significant lower area than these two works. As Virtex-2 family FPGAs are not recommended by Xilinx, we choose Spartan-3E FPGA, which has a similar structure of Virtex-2 Pro, to perform the comparison. In our implementation, only 350 Slices with 2 MULTs and 2 BRAMs are consumed, thus more than half of the resources are saved in the comparison with [22] and [21]. Roy et al. [19] recently present a single instruction based light ECC processor coupled with dedicated hardcores of the FPGAs for NIST P-256. The implementation only occupies 72 Slices, but the consumed DSP blocks and BRAMs are relatively more (8 DSP blocks and 24 BRAMs). Therefore, our implementation

achieves a good balance in the consumed logics and hardcores. Furthermore, thanks to the optimization inside DSP blocks, our ECC implementation is able to run at a high frequency and compute the ECPM for general prime fields with a low latency.

6 Conclusion

In this paper, we propose a very low-cost ECC implementation for general prime fields. In the architecture design, we efficiently integrate all the prime operations into the compact ALU, and specifically optimize its functions for elliptic curve arithmetic. In the architecture implementation, we further adopt platform-targeted optimization techniques, such as pipeline inside DSP blocks and SRL implementation in FPGAs, and this allows us to further reduce the area of the implementation and improve the efficiency. Experimental results indicate that the implementation only occupies 105 Slices, 2 DSP blocks and 2 BRAMs in Spartan-6 FPGA. In future work, we will add the countermeasures into the implementation to resist side channel attacks.

Acknowledgments. We thank the anonymous reviewers of SAC 2016 and ICICS 2016 for their invaluable suggestions and comments. This work was partially supported by National Basic Research Program of China (973 Program No. 2013CB338001), National Natural Science Foundation of China (No. 61602476, No. 61402470) and Strategy Pilot Project of Chinese Academy of Sciences (No. XDA06010702).

References

1. Bosmans, J., Roy, S.S., Järvinen, K., Verbauwhede, I.: A tiny coprocessor for elliptic curve cryptography over the 256-bit NIST prime field. In: 29th International Conference on VLSI Design and 15th International Conference on Embedded Systems, VLSID 2016, Kolkata, India, 4–8 January 2016, pp. 523–528 (2016)
2. Brier, É., Joye, M.: Weierstraß elliptic curves and side-channel attacks. In: Naccache, D., Paillier, P. (eds.) PKC 2002. LNCS, vol. 2274, pp. 335–345. Springer, Heidelberg (2002). doi:[10.1007/3-540-45664-3_24](https://doi.org/10.1007/3-540-45664-3_24)
3. McIvor, C., McLoone, M., McCanny, J.: An FPGA elliptic curve cryptographic accelerator over GF(p). In: Irish Signals and Systems Conference (ISSC), pp. 589–594 (2004)
4. Cericom Research: Standards for Efficient Cryptography - SEC-1: Elliptic curve cryptography (2000). www.secg.org/sec1-v2.pdf
5. Cericom Research: Standards for Efficient Cryptography - SEC-2: Recommended Elliptic Curve Domain Parameters (2000). www.secg.org/SEC2-Ver-1.0.pdf
6. Furbass, F., Wolkerstorfer, J.: ECC processor with low die size for rfid applications. In: International Symposium on Circuits and Systems (ISCAS) 2007, pp. 1835–1838. IEEE (2007)
7. Güneysu, T., Paar, C.: Ultra high performance ECC over NIST primes on commercial FPGAs. In: Oswald, E., Rohatgi, P. (eds.) CHES 2008. LNCS, vol. 5154, pp. 62–78. Springer, Heidelberg (2008). doi:[10.1007/978-3-540-85053-3_5](https://doi.org/10.1007/978-3-540-85053-3_5)

8. Hutter, M., Feldhofer, M., Plos, T.: An ECDSA processor for RFID authentication. In: Ors Yalcin, S.B. (ed.) *RFIDSec 2010*. LNCS, vol. 6370, pp. 189–202. Springer, Heidelberg (2010). doi:[10.1007/978-3-642-16822-2_16](https://doi.org/10.1007/978-3-642-16822-2_16)
9. Izu, T., Möller, B., Takagi, T.: Improved elliptic curve multiplication methods resistant against side channel attacks. In: Menezes, A., Sarkar, P. (eds.) *INDOCRYPT 2002*. LNCS, vol. 2551, pp. 296–313. Springer, Heidelberg (2002). doi:[10.1007/3-540-36231-2_24](https://doi.org/10.1007/3-540-36231-2_24)
10. Izu, T., Takagi, T.: A fast parallel elliptic curve multiplication resistant against side channel attacks. In: Naccache, D., Paillier, P. (eds.) *PKC 2002*. LNCS, vol. 2274, pp. 280–296. Springer, Heidelberg (2002). doi:[10.1007/3-540-45664-3_20](https://doi.org/10.1007/3-540-45664-3_20)
11. Kern, T., Feldhofer, M.: Low-resource ECDSA implementation for passive RFID tags. In: 17th IEEE International Conference on Electronics, Circuits, and Systems, ICECS 2010, Athens, Greece, 12–15. pp. 1236–1239, December 2010
12. Tawalbeh, L., Mohammad, A., Gutub, A.: Efficient FPGA implementation of a programmable architecture for $GF(p)$ elliptic curve crypto computations. *J. Sig. Process. Syst.* **59**(3), 233–244 (2010)
13. Montgomery, P.L.: Modular multiplication without trial division. *Math. Comput.* **44**(170), 519–521 (1985)
14. Montgomery, P.L.: Speeding the pollard and elliptic curve methods of factorization. *Math. Comput.* **48**(177), 243–264 (1987)
15. Guillermin, N.: A high speed coprocessor for elliptic curvescalar multiplications over F_p . *Cryptographic Hardw. Embed. Syst. (CHES)* **2010**, 48–64 (2010)
16. Office of State Commercial Cryptography Administration: Public Key Cryptographic Algorithm SM2 Based on Elliptic Curves (2012, in Chinese). <http://www.oscca.gov.cn/UpFile/2010122214822692.pdf>
17. Orup, H.: Simplifying quotient determination in high-radix modular multiplication. In: *IEEE Symposium on Computer Arithmetic*, pp. 193–199 (1995)
18. Pessl, P., Hutter, M.: Curved tags - a low-resource ECDSA implementation tailored for RFID. *Radio Freq. Ident.: Secur. Priv. Issues (RFIDSec)* **2014**, 156–172 (2014)
19. Roy, D.B., Das, P., Mukhopadhyay, D.: ECC on Your fingertips: a single instruction approach for lightweight ECC design in $GF(p)$. In: Dunkelmann, O., Keliher, L. (eds.) *SAC 2015*. LNCS, vol. 9566, pp. 161–177. Springer, Heidelberg (2016). doi:[10.1007/978-3-319-31301-6_9](https://doi.org/10.1007/978-3-319-31301-6_9)
20. Ghosh, S., Alam, M., Chowdhury, D.R., Gupta, I.S.: Parallel crypto-devices for $GF(p)$ elliptic curve multiplication resistant against side channel attacks. In: *Comput. Electr. Eng.* pp. 329–338 (2009)
21. Varchola, M., Guneyusu, T., Mischke, O.: Microecc: a lightweight reconfigurable elliptic curve crypto-processor. In: *International Conference on Reconfigurable Computing and FPGAs (ReConFig) 2011*, pp. 204–210. IEEE (2011)
22. Vliegen, J., Mentens, N., Genoe, J., Braeken, A., Kubera, S., Touhafi, A., Verbauwhede, I.: A compact fpga-based architecture for elliptic curve cryptography over prime fields. In: *International Conference on Application-Specific Systems Architectures and Processors (ASAP) 2010*, pp. 313–316. IEEE (2010)
23. Wenger, E., Feldhofer, M., Felber, N.: Low-resource hardware design of an elliptic curve processor for contactless devices. In: Chung, Y., Yung, M. (eds.) *WISA 2010*. LNCS, vol. 6513, pp. 92–106. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-17955-6_7](https://doi.org/10.1007/978-3-642-17955-6_7)
24. Ma, Y., Liu, Z., Pan, W., Jing, J.: A high-speed elliptic curve cryptographic processor for generic curves over F_p . *Sel. Areas Crypt. (SAC)* **2013**, 421–437 (2014)