An Elliptic Curve Cryptography Coprocessor over $GF(2^m)$ on a Low-Cost Embedded System

Hui Zhao, Long Wang, Guo-Qiang Bai

Institute of Microelectronics of Tsinghua University, Beijing, 100084, China

Email: zhaohui@mails.tsinghua.edu.cn

Abstract — In this paper we propose a low-cost coprocessor architecture for elliptic curves cryptography which supports the main mathematical operations for the computation of ECDSA over $GF(2^{193})$, including point doubling, point addition and scalar multiplication over field ECC. As the field is fixed, we use this special property to design special operational units and pipeline structure, only increase 4% area, but at the same time around 40% faster. Our design allows to perform a scalar multiplication over $GF(2^{193})$ in 24 ms at a clock frequency of 10 MHz, and is only 11,486 NAND gates large. The advantages of both the performance and hardware cost are much favorable compared with previous similar work.

Index Terms — Elliptic Curves Cryptography (ECC), Embedded System, hardware design, architecture

I. INTRODUCTION

Embedded systems are widely used for many different purposes in daily life, which enable profitable and legal trading, confidentiality, integrity, and non-reputability of transactions in e-business, e-government, and Internet applications. Cryptography plays a significant role in electronic world on embedded systems. Elliptic Curve Cryptography (ECC) have the highest security/ key-length ratio among all known public key and on each security level there are plenty of choices of elliptic parameters. So ECC has become most attractive for various applications.

This advantage is especially suitable for targeting embedded system as it typically provide relative limited resources, such as smaller area, lower power consumption and appropriate time-consuming calculations of scalar multiplication over field ECC.

There are numerous papers dealing with the hardware/software co-design of ECC on 8-bit CPU platforms [2, 3, 4, 6, 7, 8]. The performance of ECC coprocessors in previous work highly depended on the efficiency of the platforms, in [3, 6, 7, 8] the performance built around an AVR microcontroller is faster than that of those using 8051 [2, 4]. What's more, it's difficult to apply them to different platforms, as the differences of instructions among platforms.

However, the approach we present in this paper is different from previous work in two important aspects. First, We propose a low cost coprocessors to compute fundamental functions over $GF(2^{193})$, no matter how different the instructions of platforms are, the performance of operations over $GF(2^{193})$ maintains all the same. It can be applied to any

System-on-a-Chip (SoC) chips on 8-bit or 16-bit platforms easily. Second, as the coprocessor performs field arithmetic in a single field, we introduce some special blocks to accelerate calculations of ECC and pipeline reused structure to minimize the area. So our design has also a well-balanced tradeoff between small area and fast computation. To our knowledge, our work has the least area size among all the related work with same security level.

The paper is structured as follows: in sections II we present some basic mathematical aspects of ECC, in section III we introduce the proposed architecture and then the results and some comparisons are listed in section IV. The conclusions are drawn in section V.

II. MATHEMATICAL BACKGROUND

ECC can be commonly divided into two groups depending on the underlying field representation: prime field, $GF(p)$ and field of characteristic two, $GF(2^m)$. In this paper we choice elliptic curves based on $GF(2^m)$ which allow efficient implementations in terms of silicon area and computing time.

An elliptic curve over a field $GF(2^m)$ can be defined as the form as

$$y^2 + xy = x^3 + ax^2 + b \quad (1)$$

with $a, b \in GF(2^m)$, and $b \neq 0$.

(x, y) satisfying (1) is called a point P on the curve. The set of all points, constitute an Abelian group including point O (referred to as the “point at infinity”), the identity element.

The basic operations of ECC are point doubling and point addition within an abelian group E . Two distinct point $P, Q \in E$ can be added to $R = P + Q$, called point addition. The particular case $P + P = 2P$ is called point doubling. Performing both of those two operations involves several sub-operations, such as addition, multiplication, squaring, inversion in the underlying field $GF(2^m)$.

The main operation in any ECC over $GF(2^m)$ is scalar multiplication. The from can be demonstrated as follows: for a integer number k and a point P we can compute the point $Q = kP$.

$$Q = kP = \underbrace{P + P + \dots + P}_{k \text{ times}} \quad (2)$$

The hierarchical structure for operations in our architecture is illustrated in Fig.1.

Fig. 1. Scheme of the hierarchy for ECC

For our implementation, we use the projective coordinates based on the Montgomery Scalar Multiplication algorithm introduced in [5]. The efficiency of calculations depends largely on the efficiency of the underlying field arithmetic. We use the polynomial basis representation $(a_{m-1} \dots a_1 a_0)$ with the irreducible trinomial $F(x) = x^{193} + x^{15} + 1$.

III. COPROCESSOR ARCHITECTURE DETAILS

In this section, we describe the architecture of the ECC design working as an IP block in a top-down manner. In Section III-A, the overall system structure and the ECC coprocessor internal structure are described. Section III-B shows some commands of AU and some details of datapath. Finally, in section III-C, we describe the method to achieve field inversion.

A. System Overview

The overall system structure is illustrated in Fig. 2, which consists of three major parts: the 8-bit DW8051 microcontroller core, the ECC coprocessor unit, the memory storage unit, including Memory Management Unit (MMU), and SRAM.

Fig. 2 Block diagram of the overall system

There are lots of intermediate data which needs to store and read in the process of Montgomery Scalar Multiplication. Systems based on different microcontroller core have different performance in transforming and computing data. For example, the 8-bit platform DW8051 delivers maximum 8 data bits within 8~12 clock cycles; as a result, it spends 200~300 clock cycles in transforming the data which is 193

bits long from ECC into SRAM. More than 50% extra time is cost to transform them in the process of the scalar multiplication. That's intolerantly slow to the system.

In order to alleviate the data transformation bottleneck, we introduce the MMU block, which help the ECC coprocessor store and load data in the process of computation kP. When the ECC coprocessor works, the CPU microcontroller core stands idle. The intermediate results occurring during a scalar multiplication will be transformed through MMU between the ECC coprocessor and SRAM. In previous work [2, 3, 4, 6, 7, 8], the commands are given by the microcontroller during the process of executing the scalar multiplication to perform field arithmetic operation. The performance of scalar multiplication highly depends on the efficiency of the platforms. However in our implementation, it performs by the MU controller with a fixed finite state machine, which has nothing to do with the microcontroller when computing, leading a faster speed and independent execution.

Fig. 2 also illustrates the main internal architecture of the ECC coprocessor, which consists of three main parts: the Main Controller (MU), the arithmetic unit controller (AUC) and datapath of ECC. The MC is the ECC coprocessor's main controller as a finite state machine to conduct the AUC for computing the operations of point addition, point doubling and scalar multiplication. The AUC, also a finite state machine, controls datapath to perform the field operations.

B. Arithmetic Unit and Datapath

The AU does not only perform arithmetic operations, but also executes some reading/writing commands, including operations of loading, saving and exchanging internal registers data. Table II shows us the AUC instructions and their execution times.

TABLE II AUC INSTRUCTIONS AND EXECUTION TIMES

Command	Cycle	Description
Load	13	Load data from SRAM
MovC2A	13	Move data from reg. C to reg. A
Sav	13	Save data to SRAM
Addition	13	$C=A+B$ over infinite field
Mult	104	$C=A*B$ over infinite field
Square	1	$C=A^2$ over infinite field

Fig. 3 shows the internal architecture of datapath. It consists of four parts: the register files, adder, multiplier and square. The adder, multiplier and square are all built on with logic gates.

In order to minimize area, we utilize 432 registers to save and perform arithmetic operations. Both register files A and C are 16*13 bits large, register files B is 16*1 bits large. As the operands in ECC are much larger than the width of SRAM, we utilize a 16-bit-parallel processing, resulting in 13 cycles in load/store data. The data input of each register is connected with a multiplexer. Register files B are the simplest ones with multiplexers of two choosing one, which help load or hold

The Field Addition is the simplest of all operations, since it is a bit by bit addition, having no carry bit, which just needs 16 XOR gates to compute in 13 clock cycles.

The square over $GF(2^m)$ has a special feature:

$$(ax+b)^2=a^2x^2+2ax+b^2=ax^2+b \quad (3)$$

Set $C(x) = A^2(x)$, then,

$$C(x) = c_{192}x^{192} + c_{191}x^{191} + c_{190}x^{190} + \dots + c_2x^2 + c_1x^1 + c_0;$$

$$A(x) = a_{192}x^{192} + a_{191}x^{191} + a_{190}x^{190} + \dots + a_2x^2 + a_1x^1 + a_0;$$

$$F(x) = x^{193} + x^{15} + 1;$$

The square reduction part calculates as follows:

(4)

The square can be constructed by XOR gates with Equ. 4.

Fig. 4 shows the internal connections of square.

A bit-serial multiplier is the simplest method and needs the least area. A 193X193 multiplication is computed in 193 clock cycles. A k-digit multiplier can achieve a k-fold speedup for multiplication with the increasing the complexity of the circuit. Another trade-off between speed and area is possible by using the digit-serial multiplier. Compared to the bit-serial multiplier where only one bit of operand B is used in each iteration, here multiple bit (equal to the digit-size) of B are multiplied to the operand A in each iteration (Fig. 4). We use a digital size of 2 as it gives a good speed-up without drastically increasing the area requirement.

A 2-digit multiplier algorithm presents in Algorithm 3.1, and the circuit of the algorithm is achieved in Fig. 5.

Algorithm 3.1 field 2-digit multiplier

Input: $A = \sum_{i=0}^{192} a_i x^i, B = \sum_{i=0}^{96} B_i x^{2i} \in GF(2^m), F(x)$

Output: $C = A \bullet B$.

- ```

1. set C ← 0;
2. for i from l-1 downto 0 do
 C ← C * x2 mod F(x) + (A * Bi mod F(x))

```

### C. Field Inversion in ECC Design

So far, the implement of field inversion in hardware is the most difficult finite field operation. There are two basic types of inversion algorithms: those based on extended Euclidean algorithm and its variants, and those that use field multiplication [1]. A field inversion circuit based on extended Euclidean algorithm will add more complexity to the controller, leading to a larger chip area. Considering the design is applied to a low-cost design, we choose the latter inversion algorithm that use field square and field multiplication. Although impacting performance, the method does not add significantly to the complexity of a hardware design.

One of the most attractive inversion algorithms using field multiplication is based on the Fermat's theorem which states that in a polynomial basis:

$$a^{-1} = a^{2^m-2} = (a^{2^{m-1}-1})^2 \quad (5)$$

Table III shows the process of field inversion using the Fermat's theorem, which requires 192 field squares and 8 field multiplications, So the field inversion can be achieved through a finite state machine.

Then, we can estimate the performance of our coprocessor whether with field square or not. From [5] we can summarize the complexity of scalar multiplication over GF(2<sup>m</sup>). Noting that each operation needs data transformation, including store/load commands.

TABLE III FIELD INVERSION PROCESS

| Process                                                       | #Square    | #Mult.   |
|---------------------------------------------------------------|------------|----------|
| $x^{-1} = x^{2^{193}-2} = (x^{2^{192}-1})^2$                  | 1          | 0        |
| $x^{2^{192}-1} = (x^{2^{96}-1})^{2^{96}} \cdot x^{2^{96}-1}$  | 96         | 1        |
| $x^{2^{96}-1} = (x^{2^{48}-1})^{2^{48}} \cdot x^{2^{48}-1}$   | 48         | 1        |
| $x^{2^{48}-1} = (x^{2^{24}-1})^{2^{24}} \cdot x^{2^{24}-1}$   | 24         | 1        |
| $x^{2^{24}-1} = (x^{2^{12}-1})^{2^{12}} \cdot x^{2^{12}-1}$   | 12         | 1        |
| $x^{2^{12}-1} = (x^{2^6-1})^{2^6} \cdot x^{2^6-1}$            | 6          | 1        |
| $x^{2^6-1} = (x^{2^3-1})^{2^3} \cdot x^{2^3-1}$               | 3          | 1        |
| $x^{2^3-1} = (x^{2^2-1})^2 \cdot x$                           | 1          | 1        |
| $x^{2^2-1} = (x^{2^1-1})^{2^1} \cdot x^{2^1-1} = x^2 \cdot x$ | 1          | 1        |
| <b>Total</b>                                                  | <b>192</b> | <b>8</b> |

TABLE IV EVALUATION OF SCALAR MULTIPLICATION

|              | Numbers   | Cycles <sup>1</sup> | Cycles <sup>2</sup> |
|--------------|-----------|---------------------|---------------------|
| #Square      | 5(m-1)+3  | 26,001              | 125,190             |
| #Mult.       | 6(m-1)+10 | 151,060             | 151,060             |
| #Add.        | 3(m-1)+6  | 22,698              | 22,698              |
| #Inv.        | 1         | 6,224               | 26,000              |
| <b>Total</b> | -----     | <b>205,983</b>      | <b>32,4948</b>      |

<sup>1</sup>: The computation with field square

<sup>2</sup>: The computation without field square

Every operation just requires extra 26 clock cycles to initialization. It illuminates us that square accelerates the performance of scalar multiplication over 40% faster.

### IV. RESULTS AND COMPARISONS

The ECC coprocessor has been synthesized in 0.18  $\mu m$  CMOS technology (VLSI SMIC18 library) by means of the Synopsys Design Compiler. We set the targeted delay for the critical path to 17 ns, as the low-cost microcontroller works at 50MHz frequency at most. In table V we summarized the figures of merit of the synthesis.

In the datapath unit which contributes nearly 73% area, square, multiplier and registers contain 375 gates, 1,455 gates and 3,102 gates correspondingly. Our design only increases 4% area of the total design for square, almost 380 NAND gates, but at the same time around 40% faster, showing a drastic speed-up using field square.

TABLE V SYNTHESIS CHIP AREA

| Component     | Sub-unit  | Size/ $\mu m^2$ | Size/GE |
|---------------|-----------|-----------------|---------|
| <b>DW8051</b> | 8051 core | 109,769         | 11,000  |
|               | 8051 IRAM | 27,941          | 2,800   |
| <b>MMU</b>    | MMU       | 1,357           | 136     |
| <b>SRAM</b>   | SRAM      |                 |         |
| <b>ECC</b>    | MU        | 19,682          | 1,972   |
|               | AUC       | 11,625          | 1,165   |
|               | Datapath  | 83,230          | 8,341   |
| <b>Total</b>  | ALL       | 253,604         | 25,414  |

TABLE VI  
COMPARISON WITH ECC SCALAR MULTIPLICATION OF  
RELATED WORK

| Ref.        | Target System | Security Level | Freq MHz | Comparison |        |
|-------------|---------------|----------------|----------|------------|--------|
|             |               |                |          | Area/ GE   | Cycles |
| [3]         | ARM9          | 163 bit        | 44.0     | 46.0k      | 134.1k |
| [7]         | AVR           | 192 bit        | 10.00    | -----      | 450.0k |
| [4]         | 8051          | 191 bit        | 10.00    | 25.0k      | 341.4k |
| [2]         | 8051          | 191 bit        | 12.00    | 12.7k      | 1.41M  |
| <b>Ours</b> | 8051          | 193 bit        | 50.00    | 11.5k      | 240.0k |

We post-stimulate our coprocessor with ModelSim SE 6.1b. It requires 239,000 clock cycles to calculate a scalar multiplication. Table VI compares the performances of ECC scalar multiplication performance with related work. Among all the performances and hardware cost of ECC coprocessor, our work with the same security level needs least silicon area than others, and at the most cases, it has a faster computation speed. Thanks to MMU which allows direct memory access, it contributes to the efficient data transfer between ECC coprocessor and SRAM. There is an exception that related

work [3] performs faster. However, it's around four times as large as ours.

## VII. CONCLUSION

This paper presents a low-cost architecture for elliptic curve cryptography and an embedded system. Compared with other related work, this coprocessor isn't strongly dependence on the efficiency of platform and can work as IP block to different embedded system much easier. We also introduce MMU to help the ECC coprocessor store and load data in the process of computation kP, which does not only speed up the performance of the ECC coprocessor, but also make coprocessor work independently, having nothing to do with the microcontroller.

The ECC coprocessor using Montgomery Scalar Multiplication requires 240,000 cycles. Result in synthesis operating frequency is 50 MHz and gate counts are approximately 11,500. Although increasing extra area as a specific square on a silicon chip, 4% more area, it has much higher performance in computation, about 40% faster than before. To our knowledge, it's the coprocessor which is the smallest area size among other ECC low-cost implements with the same security level. In addition, it has a relatively fast performance. All of these features make our work more favorable than previous work on low-cost embedded systems.

## ACKNOWLEDGEMENT

The research described in this paper was supported by .

## REFERENCES

- [1] Darrel Hankerson, Alfred Menezes and Scott Vanstone, "Guide to Elliptic Curve Cryptography" *Springer-Verlag New York, Inc.*, 2004.
- [2] M. Koschuch, J. Lechner, A. Weitzer, J. Großschädl, A. Szekely, S. Tillich, and J. Wolkerstorfer, "Hardware/Software Co-Design of Elliptic Curve Cryptography on an 8051 Microcontroller," *CHES 2006, LNCS 4249*, pp. 430-444, Springer Verlag, 2006.
- [3] Jin Park, Jeong-Tae Hwang and Young-Chul Kim, "FPGA and ASIC Implementation of ECC Processor for Security on Medical Embedded System", *ICITA '05*, IEEE.
- [4] Harald Aigner, Holger Bock, and Johannes Wolkerstorfer, "A Low-Cost ECC Coprocessor for Smartcards", *CHES 2004, LNCS 3156*, pp. 107-118, Springer Verlag, 2004.
- [5] Julio Lopez and Ricardo Dahab, "Fast Multiplication on Elliptic Curves over  $GF(2^m)$  without Precomputation", *CHES 1999, LNCS 1717*, pp. 316-327, Springer Verlag, 1999.
- [6] Sandeep Kumar and Christof Paar, "Reconfigurable Instruction Set Extension for Enabling ECC on an 8-bit Processor", *FPL 2003, LNCS 320*, pp. 586-596, Springer Verlag, 2004.
- [7] S. Janssens, J. Thomas, W. Borremans et al, "Hardware/software co-design of an elliptic curve public-key cryptosystem", *SIPS 2001*, pp. 209-216. IEEE, 2001.
- [8] Hans Eberle, Arvinderpal Wander, Nils Gura, et al, "Architectural Extensions for Elliptic Curve Cryptography over  $GF(2^m)$  on 8-bit Microprocessors  $GF(2^m)$ ", *ASAP 2005*, IEEE Computer Society Press, 2005.