A SOFT-CORE PROCESSOR FOR FINITE FIELD ARITHMETIC WITH A VARIABLE WORD SIZE ACCELERATOR

Aiko Iwasaki, Keisuke Dohi, Yuichiro Shibata, Kiyoshi Oguri, Ryuichi Harasawa

Graduate School of Engineering, Nagasaki University
1-14, Bunkyo, Nagasaki, Japan
email: {iwasaki,dohi}@pca.cis.nagasaki-u.ac.jp,{shibata,oguri,harasawa}@cis.nagasaki-u.ac.jp

ABSTRACT

This paper presents implementation and evaluation of an accelerator architecture for soft-cores to speed up reduction process for the arithmetic on $GF(2^m)$ used in Elliptic Curve Cryptography (ECC) systems. In this architecture, the word size of the accelerator can be customized when the architecture is configured on an FPGA. Focusing on the fact that the number of the reduction processing operations on $GF(2^m)$ is affected by the irreducible polynomial and the word size, we propose to employ an unconventional word size for the accelerator depending on a given irreducible polynomial and implement a MIPS-based soft-core processor coupled with a variable-word size accelerator. As a result of evaluation with several polynomials, it was shown that the performance improvement of up to 10.2 times was obtained compared to the 32-bit word size, even taking into account the maximum frequency degradation of 20.4 % caused by changing the word size. The advantage of using unconventional word sizes was also shown, suggesting the promise of this approach for low-power ECC systems.

1. INTRODUCTION

In this paper, we focus on the algorithm of reduction processing [1] on finite fields $GF(2^m)$ of characteristic 2, which is one of the most time consuming basic operations in elliptic curve arithmetic. Scott proposed a technique to choice optimal irreducible polynomials for a given architecture [2]. In contrast, we propose to use a soft-core processor with configurable word size for ECC arithmetic and to tailor the architecture to a given irreducible polynomial, by making the best use of flexibility of FPGAs. Compared to Scott's approach, our approach allows us to use standard irreducible polynomials widely used in many ECC applications [3]. While many researches to speed up finite field arithmetic for ECC using FPGAs have been reported so far [4][5][6][7][8][9], there have been hardly any attempts to implement a soft-core processor with configurable ECC arithmetic accelerator with variable word size, as far as the authors' knowledge goes.

2. MATHEMATICAL BACKGROUND

ECC which was proposed by Koblitz[10] and Miller[11] in 1985, has the advantage that smaller key size can be used compared to RSA of equivalent cryptographic strength. For ECC, we often use elliptic curves defined over $GF(2^m)$ from an implementational point of view, where $GF(2^m)$ is an extension field of degree m over $GF(2)$. The elements in $GF(2^m)$ are expressed with m-bit binary numbers. While addition and subtraction on $GF(2^m)$ can be realized as an m-bit XOR operation without carry propagation, multiplication needs to be followed by reduction processing, which is an operation to obtain a remainder of an intermediate product divided by the irreducible polynomial of the field and tends to be time consuming.

2.1. Reduction processing

We express the field $GF(2^m)$ as the residue class ring as $GF(2)[x]/(f(x))$, where $f(x)$ is an irreducible polynomial over $GF(2)$ of degree m. When $f(x)$ is of the form

$$f(x) = x^m + x^a + x^b + x^c + 1 \tag{1}$$

with $m > a > b > c > 0$, we see

$$x^m = x^a + x^b + x^c + 1 \tag{2}$$

because $f(x) = 0$ in $GF(2^m)$. By iteratively substituting Eq. (2), any polynomials can be reduced so as to have a smaller degree than that of the irreducible polynomial.

For example, let $g(x)$ be a polynomial of degree $2m-2$. By Eq. (2), we have

$$\begin{aligned} g(x) &= g_{2m-2}x^{2m-2} + \cdots + g_m x^m + g_{m-1}x^{m-1} \\ &\quad + \cdots + g_1 x + g_0 \\ &= (g_{2m-2}x^{m-2} + \cdots + g_m)(x^a + x^b + x^c + 1) \\ &\quad + g_{m-1}x^{m-1} + \cdots + g_1 x + g_0 \end{aligned} \tag{3}$$

where $g_i \in GF(2) = \{0, 1\}$. Namely, the term $x^i x^m$ is reduced to the i-bit left shift result of the binary representation of $r(x) := x^a + x^b + x^c + 1$. So, the reduction process can

be performed by checking each bit of $g(x)$ from the MSB and shift-and-xoring $r(x)$ into $g(x)$ when the corresponding bit is 1.

2.2. Fast reduction processing algorithm

In software processing, the speed of the reduction algorithm can be improved by using word size operations [1]. For example, let us assume that the reduction by $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ is performed on a machine with a 32-bit word length. As shown in Fig. 1, polynomials are represented as a bit string that spans on multiple 32-bit words. Most operations in this algorithms are shift and XOR, and the number of variations of shift amounts appearing in the code is only eight as shown in Table 1. Shift amounts required for the algorithm are given by the formulas shown in Table 2 when the word size is ω.

2.3. Effect of processing word size

The operation count of the reduction algorithm largely depends on a processing word size. First of all, the longer the processing word size becomes, the more parallelism can be extracted from long-word XOR operations. In addition, since the required shift amounts depend on the relationship between the formula of the irreducible polynomial and word size, the required number of shift operations can be decreased by selecting a suitable processing word size for a given irreducible polynomial.

Fig. 2 demonstrates how the word size impacts the reduction process shown in Fig. 1. Obviously, there are some special word sizes that significantly reduce the required operations counts compared to neighbor sizes by canceling out many shift operations. This motivated us to take the FPGA implementation approach that makes possible to select even an unconventional word size.

3. DESIGN AND IMPLEMENTATION

Since ECC is a basic building block for constructing a variable public key cryptosystem protocols, it is desirable that the system offers some sort of software programability. Therefore, we propose an architecture of an FPGA-based soft-core processor coupled with an accelerator for ECC arithmetic as shown in Fig. 3.

The main processor is based on the MIPS architecture[12] and offers standard functionality for software execution. The accelerator has the same pipeline structure to the MIPS and is connected to the main core at the instruction decode and register fetch (Id) and memory access (Ma) stages. This accelerator has its own register file (ECC_regfile) and arithmetic circuit (ECC_ALU). The colored datapath in Fig. 3 can be configure to any designated word size. The word

Algorithm reduction processing modulo $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$

Input: A binary polynomial represented as 18 32-bit words $g[.]$
Output: $g[.]$ modulo $f(x)$, represented as 9 32-bit words

1: $g[17] \leftarrow g[17], g_{16} \leftarrow g[16], g_{15} \leftarrow g[15], g_{14} \leftarrow g[14], g_{13} \leftarrow g[13], g_{12} \leftarrow g[12], g_{11} \leftarrow g[11], g_{10} \leftarrow g[10], g_9 \leftarrow g[9], g_8 \leftarrow g[8], g_7 \leftarrow g[7], g_6 \leftarrow g[6], g_5 \leftarrow g[5], g_4 \leftarrow g[4], g_3 \leftarrow g[3], g_2 \leftarrow g[2], g_1 \leftarrow g[1], g_0 \leftarrow g[0]$
2: $g[17] \leftarrow g[16] \leftarrow g[15] \leftarrow g[14] \leftarrow g[13] \leftarrow g[12] \leftarrow g[11] \leftarrow g[10] \leftarrow g[9] \leftarrow 0$
3: $g_9 = g_9 \oplus (g_{17} \gg 15) \oplus (g_{17} \gg 20) \oplus (g_{17} \gg 22) \oplus (g_{17} \gg 27)$
4: $g_8 = g_8 \oplus (g_{17} \ll 17) \oplus (g_{17} \ll 12) \oplus (g_{17} \ll 10) \oplus (g_{17} \ll 5) \oplus (g_{16} \gg 15) \oplus (g_{16} \gg 20) \oplus (g_{16} \gg 22) \oplus (g_{16} \gg 27)$
5: $g[7] = g_7 \oplus (g_{16} \ll 17) \oplus (g_{16} \ll 12) \oplus (g_{16} \ll 10) \oplus (g_{16} \ll 5) \oplus (g_{15} \gg 15) \oplus (g_{15} \gg 20) \oplus (g_{15} \gg 22) \oplus (g_{15} \gg 27)$
6: $g[6] = g_6 \oplus (g_{15} \ll 17) \oplus (g_{15} \ll 12) \oplus (g_{15} \ll 10) \oplus (g_{15} \ll 5) \oplus (g_{14} \gg 15) \oplus (g_{14} \gg 20) \oplus (g_{14} \gg 22) \oplus (g_{14} \gg 27)$
7: $g[5] = g_5 \oplus (g_{14} \ll 17) \oplus (g_{14} \ll 12) \oplus (g_{14} \ll 10) \oplus (g_{14} \ll 5) \oplus (g_{13} \gg 15) \oplus (g_{13} \gg 20) \oplus (g_{13} \gg 22) \oplus (g_{13} \gg 27)$
8: $g[4] = g_4 \oplus (g_{13} \ll 17) \oplus (g_{13} \ll 12) \oplus (g_{13} \ll 10) \oplus (g_{13} \ll 5) \oplus (g_{12} \gg 15) \oplus (g_{12} \gg 20) \oplus (g_{12} \gg 22) \oplus (g_{12} \gg 27)$
9: $g[3] = g_3 \oplus (g_{12} \ll 17) \oplus (g_{12} \ll 12) \oplus (g_{12} \ll 10) \oplus (g_{12} \ll 5) \oplus (g_{11} \gg 15) \oplus (g_{11} \gg 20) \oplus (g_{11} \gg 22) \oplus (g_{11} \gg 27)$
10: $g[2] = g_2 \oplus (g_{11} \ll 17) \oplus (g_{11} \ll 12) \oplus (g_{11} \ll 10) \oplus (g_{11} \ll 5) \oplus (g_{10} \gg 15) \oplus (g_{10} \gg 20) \oplus (g_{10} \gg 22) \oplus (g_{10} \gg 27)$
11: $g[1] = g_1 \oplus (g_{10} \ll 17) \oplus (g_{10} \ll 12) \oplus (g_{10} \ll 10) \oplus (g_{10} \ll 5) \oplus (g_9 \gg 15) \oplus (g_9 \gg 20) \oplus (g_9 \gg 22) \oplus (g_9 \gg 27)$
12: $g_0 = g_0 \oplus (g_9 \ll 17) \oplus (g_9 \ll 12) \oplus (g_9 \ll 10) \oplus (g_9 \ll 5)$
13: $t = (g_8 \gg 27)$
14: $g_0 = g_0 \oplus t$
15: $t = (t \ll 27)$
16: $g[8] = g_8 \oplus t$
17: $g[0] = g_0 \oplus (t \gg 15) \oplus (t \gg 20) \oplus (t \gg 22)$

Figure 1: Reduction algorithm by $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$

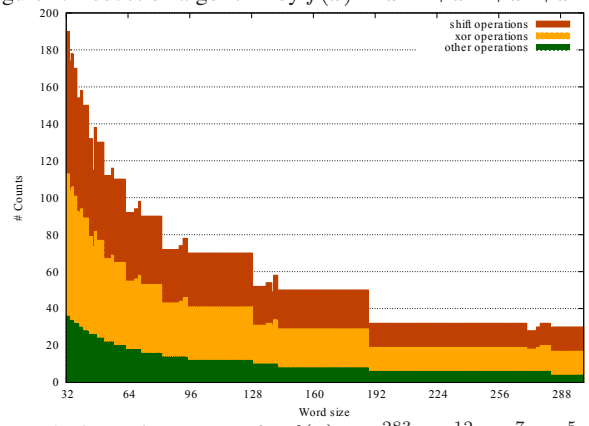

Figure 2: Operations counts for $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$

Table 1: Shift amounts for the algorithm in Fig. 1

	right shift amount	left shift amount
x^{12}	15	17
x^7	20	12
x^5	22	10
1	27	5

Table 2: Shift amounts for the reduction algorithm

	right shift amount	left shift amount
x^a	$m - a \pmod{\omega}$	$\omega - (m - a \pmod{\omega})$
x^b	$m - b \pmod{\omega}$	$\omega - (m - b \pmod{\omega})$
x^c	$m - c \pmod{\omega}$	$\omega - (m - c \pmod{\omega})$
1	$m \pmod{\omega}$	$\omega - (m \pmod{\omega})$

size of the ECC accelerator can be configured to a given irreducible polynomial for the application system. The main memory of the proposed soft-core is provided by on-chip BRAM. The architecture was designed in Verilog-HDL RTL descriptions.

3.1. Architecture of the accelerator

In the ECC accelerator, the dedicated ALU provides XOR and shift operations for the reduction process. In general, shift hardware for large word size tends to consume huge logic area and become a bottleneck for the clock speed. Fortunately, however, reduction process requires only a few types of shift amounts according to a given irreducible polynomial and word size as aforementioned. This allows us to implement significantly simple and fast shift hardware for long word size. The ECC register file provides 32 registers of the designated word size, where the value of the register zero is always 0. The word size for ECC arithmetic is given as a parameter at the stage of logic synthesis.

3.2. Data memory alignment

While the word size for the ECC accelerator can be tailored to any length, that of the MIPS main core is fixed to be 32 bits. Therefore, memory space sharing between the accelerator and the main core causes a complex alignment issue.

In order to make the architecture simple and efficient, we put some alignment constraints on memory access made by the ECC accelerator. When the word size w is designated by a designer, the main memory is automatically configured with BRAM to have the data width $w_d = \max\left(2^{\lceil \log_2(w) \rceil}, 32\right)$. Here, the value of 32 comes from the word size of the MIPS main core. Since always $w_d \geq w$, load and store instructions for the ECC registers are executed in one clock cycle. However, in order to avoid introducing a lot of multiplexers, effective address of ECC load/store must be aligned to a multiple of w_d. Therefore, when ECC accelerator access to the main data memory, the accelerator can not access to the upper region of $w_d - w$ bits of the main memory word. Effective addresses of ECC load/store instructs are calculated in the MIPS main-core processor.

3.3. C library

We also developed a C library to embed dedicated ECC instructions in C source code using an in-line assembler. Users can develop applications using a GCC MIPS cross compiler with this library. Our ECC dedicated instructions can be invoked as a function call with operand registers as arguments. At present, we support the functions using in reduction processing.

4. EVALUATION

We used a Xilinx Spartan-6 XC6SLX45 FPGA as a target device and implemented functions of the reduction processing with three irreducible polynomials, $f(x) = x^{241} + x^{70} + 1$, $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$ and $f(x) = x^{163} + x^7 + x^6 + x^3 + 1$. The evaluated code was compiled by an MIPS cross compiler based on GNU Compiler Collection (GCC) with our function library for the ECC dedicated instructions. In this time, we used GNU binutils-2.23.2 and gcc-core-4.2.4 in the little-endian mode.

First, we evaluated how the increase in the word size restricts the maximum operational clock frequency. We mapped every architectural configuration of our architecture, changing the ECC word size from 32 bits to 300 bits and performed static timing analysis. Fig. 4 shows the frequency degradation caused by the increase in the ECC word size was not so severe; approximately 0.03 MHz per 1 bit in average. This is due to a long-word XOR instruction does not cause carry propagation and a long-word shift for ECC does not increase the number of potential shift amounts. The impact on the frequency given by irreducible polynomials was slight, since the hardware differences are just shift amounts. There are relatively large frequency drops at 128 bit and 256 bit. This is due to additional multiplexers were inserted between the main core and the long-word main memory bank.

Next, we evaluated how the number of clock cycles required for the reduction processing was influenced by changing the ECC word size. From results of Fig. 5, by selecting an appropriate word size for the given irreducible polynomial, the required clock cycles were significantly reduced and this effect was obviously superior the frequency drop shown in Fig. 4.

Fig. 6 demonstrates how the proposed architecture improved the execution performance of the reduction processing compared to the 32-bit MIPS main-core processor. As a result, the performance was improved by up to 10.2 times in spite of the maximum frequency degradation of 20.4 %. In addition, the evaluation results clearly reveal the advantage of employing an unconventional word size for the ECC arithmetic, which is not a power of two number. Meanwhile, increase in FPGA resources by adding the accelerator was 2.84 times and 3.79 times for FFs and LUTs, on the 294-bit architecture for $f(x) = x^{283} + x^{12} + x^7 + x^5 + 1$. Considering the performance enhancement obtained, this increase in resources would be reasonable.

5. CONCLUSION

This paper presented implementation of an accelerator with variable word size for a MIPS-based soft-core to speed up reduction processing on $GF(2^m)$ arithmetic widely used in ECC systems. In our architecture, the word size for ECC arithmetic can be given as a design parameter in Verilog-

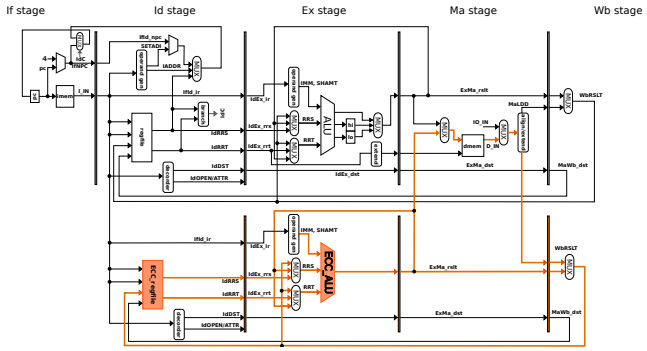

Figure 3: Overview of proposed design

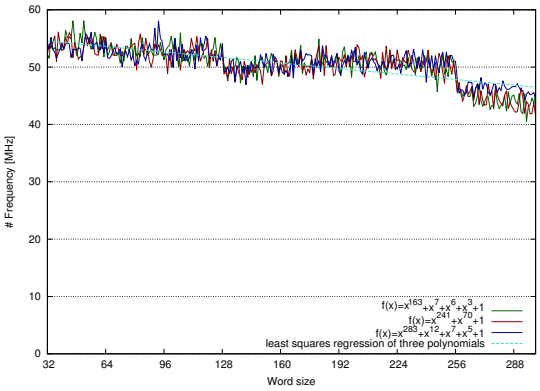

Figure 4: Maximum clock frequency and word size

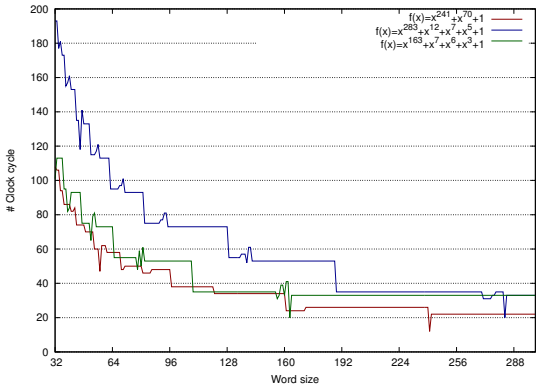

Figure 5: Required clock cycles and word size

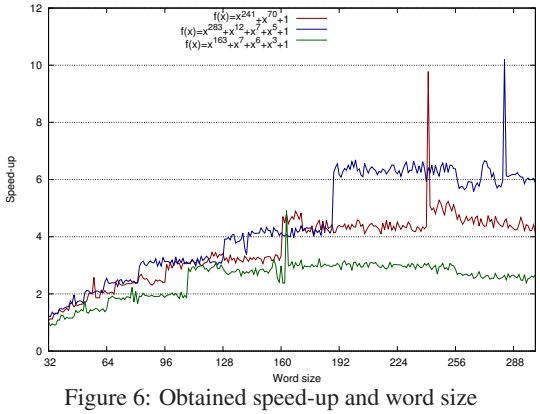

Figure 6: Obtained speed-up and word size

HDL description, so that optimal word size can be selected according for the irreducible polynomial. We also developed a function library, so that users can easily insert dedicated ECC instructions in C source code. Evaluation results showed that although the maximum clock frequency was degraded according to an increase in word size, the performance was improved by up to 10.2 times by choosing the best word size for a given irreducible polynomials. In addition, it was shown that adopting an unconventional word size such as 294 bits by making the best use of flexibility of FPGAs was significantly advantageous.

6. REFERENCES

[1] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curves Cryptography*. Springer, 2004, pp. 53–56.

[2] M. Scott, "Optimal irreducible polynomials for $GF(2^m)$ arithmetic," in *Cryptology ePrint Archive, Report 2007/197*, 2007.

[3] D. Brown, "Sec 2: Recommended elliptic curve domain parameters," in *Standards for Efficient Cryptography, ver. 2.0*. Certicom Corup, 2010.

[4] K. O. Takehiro Ito, Yuichiro Shibata, "Implementation of the extended euclidean algorithm for the tate pairing on fpga," in *Field-Programmable Logic and Applications*. Springer, 2004.

[5] J.-J. Q. Guerric Meurice de Dormale, Philippe Bulens, "Efficient modular division implementation, ecc over $GF(p)$ affine coordinates application," in *Field-Programmable Logic and Applications*. Springer, 2004.

[6] C. P. Sandeep Kummar, "Reconfigurable instruction set extension for enabling ecc on an 8-bit processor," in *Field-Programmable Logic and Applications*. Springer, 2004.

[7] W. M. Maurice Keller, Tim Kerins, "FPGA implementation of a $GF(2^m)$ multiplier for use in pairing based cryptosystems," in *FPL 2005*, 2005.

[8] H. Lia, J. Huangb, P. Sweanya, and D. Huang, *FPGA implementations of elliptic curve cryptography and Tate pairing over a binary field*. Journal of Systems Architecture, 2008, pp. 1077–1088.

[9] M. Keller, R. Ronan, W. Marnane, and C. Murphy, *Hardware architectures for the Tate pairing over $GF(2^m)$*. Computers and Electrical Engineering, 2007, pp. 392–406.

[10] N. Koblitz, *Elliptic curve cryptosystems*. American Mathematical Society, 1987, pp. 203–209.

[11] V. S. Miller, *Use of Elliptic Curves in Cryptography*. Springer Berlin Heidelberg, 1986, pp. 417–426.

[12] D. A. Patterson and J. L. Hennessy, *COMPUTER ORGANIZATION AND DESIGN THE HARDWARE / SOFTWARE INTERFACE*, 4th ed. Morgan Kaufmann, 2009.