Implementation of ECDSA in Combo6X Card

Tomáš Davidovič
CTU in Prague, FEE, Department of Computer Science and Engineering,
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
davidt2@fel.cvut.cz

Martin Havlan
CTU in Prague, FEE, Department of Telecommunications,
Technická 2, 166 27 Praha 6, Czech Republic
havlan@fel.cvut.cz

Martin Novotný
CTU in Prague, FEE, Department of Computer Science and Engineering,
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
novotnym@fel.cvut.cz

Jan Schmidt
CTU in Prague, FEE, Department of Computer Science and Engineering,
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
schmidt@fel.cvut.cz

Abstract. *This paper describes implementation of a cryptographic coprocessor performing operations on elliptic curve points with affine coordinates in GF(2^m). The coprocessor will be implemented in Xilinx Virtex II Pro XC2VP50 on a PCI-X card Combo6X. The goal is to compare the performance of solutions based on polynomial and normal bases when implemented into a complex system. Because of this, the coprocessor has interchangeable arithmetic units and a microprogrammable controller, allowing easy switching between both architectures. The paper describes the basics of the used algorithms, their implementation and proposes measurement techniques for comparing the effectiveness of both variants in the context of a PC system.*

Keywords: ECDSA, Elliptic Curve Cryptography, VHDL, coprocessor, Combo6X, FPGA

1. Introduction

While Elliptic Curve Digital Signature Algorithm (ECDSA) with affine point coordinates in *GF*(2^m) has been standardized [4], there are still options with considerable degree of freedom. Our task is to implement a coprocessor that will help us decide whether, for a specific key length, normal basis performs better than polynomial basis.

Addition and subtraction of two elements is the same in both bases and is implemented as a simple bitwise XOR operation.

Squaring, multiplication and division are specific for each respective basis, but can be viewed as blackboxes and incorporated as arithmetic units into a universal coprocessor, using a common interface.

We will first describe the used, and proposed, algorithms and their modifications. Then we will describe the current coprocessor architecture. Finally we propose some measurement techniques.

2. Algorithms

The desired result obtained from the coprocessor is a scalar multiple of a point: $k*P = P+P+...+P$ (k-times).

We will use Horner scheme (add-and-double algorithm) to evaluate the scalar point multiple. For this we need to imple-

ment two operations on elliptic curve points, addition of two points ($P+Q$) and point doubling ($2*P=P+P$).

Each point is characterized by its coordinates and, regardless of the coordinate system used, the coordinates are elements of $GF(2^m)$. All point operations can therefore be expressed using the following field operations: addition, multiplication, division (or inversion) and squaring [4].

As already stated, addition and subtraction are the same for both polynomial and normal bases and are a simple bitwise XOR.

While squaring can be performed using multiplication, there is a simpler, single cycle, squaring algorithm in both bases. Squaring in polynomial basis consists of "spreading" the polynomial to double length, with zeros at each odd position and then reduction modulo the field polynomial. For small field polynomials (trinomials and pentanomials) the reduction can be done using a small number of XOR gates for each bit, usually two or three gates for the most typical values of m. The XOR gate mesh is generated separately and is unique for each m and each field polynomial. In normal basis, squaring is performed by simple cyclic-shift (rotation).

Multiplication in normal basis is computed using pipelined bit-serial Massey-Omura [6] multiplier. It also allows simultaneous processing of D bits, referenced as a "digit".

The polynomial basis multiplier, evaluating the expression $C = A \times B \bmod F(x)$ (where $F(x)$ is the field polynomial), is currently a simple LSB (least significant bit) multiplier. Further research is being made into possibilities of implementing simultaneous processing of D bits in polynomial basis as well, to allow better scalability of the design.

The division is generally performed as multiplication of the dividend and the inverse value of the divisor.

In normal basis the Itoh, Teechai, Tsujii [5] algorithm is used for inversion of the divider. The ITT algorithm is based on Fermat's little theorem which states, that $a^p=a \bmod p$, therefore $a^{p-2} = a^{-1} \bmod p$. This principle can be used for any basis, but the extended Euclidean algorithm (EEA) proved to be more efficient when it can be used.

In polynomial basis we therefore use the EEA that also allows immediate division instead of just inversion.

The algorithm for point addition and point doubling is shown on Fig. 1.

Output: the point $P_2 := P_0 + P_1$.

1. If $P_0 = \circ$, then output $P_2 \leftarrow P_1$ and stop
2. If $P_1 = \circ$, then output $P_2 \leftarrow P_0$ and stop
3. If $x_0 \neq x_1$ then
   - 3.1 set $\lambda \leftarrow (y_0 + y_1) / (x_0 + x_1)$
   - 3.2 set $x_2 \leftarrow a + \lambda^2 + \lambda + x_0 + x_1$
   - 3.3 go to step 7
4. If $y_0 \neq y_1$ then output $P_2 \leftarrow \circ$ and stop
5. If $x_1 = 0$ then output $P_2 \leftarrow \circ$ and stop
6. Set
   - 6.1 $\lambda \leftarrow x_1 + y_1 / x_1$
   - 6.2 $x_2 \leftarrow a + \lambda^2 + \lambda$
7. $y_2 \leftarrow (x_1 + x_2)\,\lambda + x_2 + y_1$
8. $P_2 \leftarrow (x_2, y_2)$

Figure 1. Addition of two points

3. Architecture

The current coprocessor design [2] uses the data structure shown on Fig. 2. It is designed to implement the algorithm described on Fig. 1.

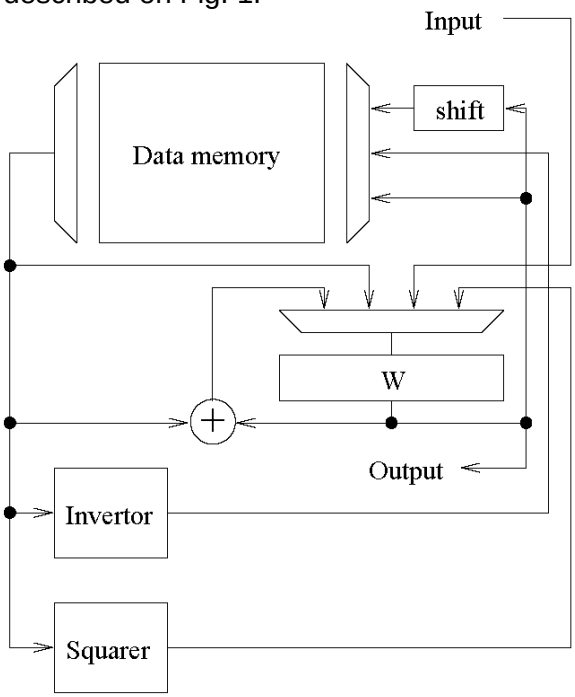

Figure 2. Main architecture data path

All but two data paths have m-bit width; the width of Input and Output paths

depends on width of the interface. Currently the width is 32 bits, the standard PCI is used. We plan to extend the width to 64 bits for use with PCI-X (PCI-64).

The work register W is implemented as a shift register, shifting its content by 16 bits, to allow input and output of data of variable length via PCI interface. The other 16 bits of 32 bit interface are used as address and various control signals for the processor. Because the Combo6X PCI interface is different from the proposed PCI interface and it uses separate buses for data and address, the work register W will be changed to shift by 32, respectively 64, bits, depending on the concrete PC architecture used.

The *Inverter* block here represents both inversion and multiplication units, because in both normal and polynomial basis the units are unified, doing just slightly different operations over the same set of registers.

Data memory is represented by block RAMs and contains 32 *m*-bit registers. Four of them are used for point coordinates, one is used for the scalar multiplicand *k*, and the rest is used for intermediate results and reserved for future use.

We will need to use each of *k*'s bits for the add-and-double algorithm. It was decided to use just *k*'s MSB and implement logical rotation in one of the paths leading to the data memory.

The whole architecture is designed to be universal, allowing not only affine coordinates, but also projective coordinates. To make the coprocessor as universal as possible, there is also a microprogrammable controller that also allows easy change of the coprocessor's behavior without any need to re-synthesize the whole design.

It is possible that the architecture could be finetuned to perform the algorithm faster, but possibly at the loss of universality. This possibility is also being researched.

4. Combo6X

Combo6X card, developed by Masaryk University and TU Brno, has been chosen as our target platform because it offers all the required resources and tools without the necessity of a complicated hardware and software development. There is also a wide range of networking applications (including, but not limited to, fast network cards and hardware routers) and the crypto coprocessor can be easily incorporated to provide support for an automatic or semi-automatic key exchange, should it be desired. Current version of the card supports PCI-X, but Combo6E, supporting PCI-Express, is being developed and will offer even higher transfer speeds with minimal changes to the current designs.

Interface between PC and the card is very straightforward. After the initialization, we can simply read (and write) to any address of any Combo6-family card in the computer. We can read either a single 32-bit word, or an array of four 32-bit words. The first can be used to check whether the processing has already finished, while the later will be used for transfer of wide data values.

On the card side of the interface, there are several components that offer a simple way to pass data along the PCI. Normally the transfer runs at 100MHz, with 16-bits per clock cycle, which gives us transfer speed 1.6Gbps. With 160 bit keys this would mean 500ns of transfer per an evaluated key with the expected processing time about 40 µs. Should this prove to be too slow, it is also possible to speed up the transfer at a cost of losing some of the interface universality.

The initial design will use the current PCI transfer components and modifications will be considered based on the test results.

5. Polynomial Digit Multiplier

Probably the biggest problem of the current architecture is that the polynomial multiplier is purely bit-serial. This wasn't a big issue when the design was tested only by a post-place-and-route simulation, where it could be simply run at the highest safe frequency. However, in a real hardware system we are limited by frequencies available from the crystal. Namely for Combo6X it is 50, 100 and 125MHz, with 100MHz being the most likely working frequency. If we chose a work frequency different from the transfer one, we would need to solve problems associated with a dual-clock design.

It is desirable to have the design as scalable as possible. The scalability allows us to maximize the throughput while keeping time constraints given by the chosen frequency.

It has been proposed to use a different polynomial multiplier. Instead of the bit-serial multiplier a digit-serial (also referenced as bit-parallel) multiplier will be used.

The principle of a digit-serial multiplier is similar to that of a LSB multiplier, but instead of multiplying by the least significant bit we will multiply by the least significant digit. Therefore we will divide the operand *B* into *D*-bit wide digits, padding the most significant bits of the operand with zeros if necessary.

To get the result of a single digit multiplication, we add together results of AND between *N*th bit of the lowest multiplier digit and operand *A* shifted left by *N* bits, where $N \in \langle 0, D\text{-}1 \rangle$. The result can be seen on Fig. 3., where b_{Di+N} denotes *N*th bit of the *i*-th lowest digit, black dots represent ANDs and grey bars represent bits to be added (XORed) together. [3]

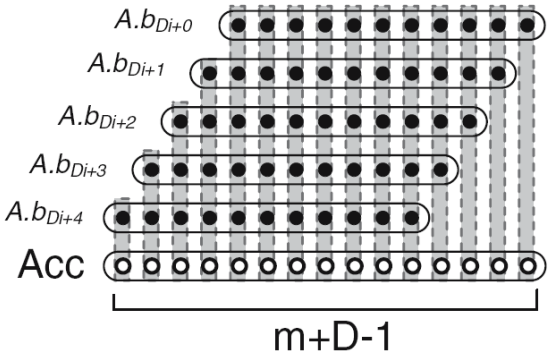

Figure 3. Multiple digit adder core

The results of each single digit multiplication are added to an accumulator that has *m*+*D*-1 bits. When the whole multiplication is finished, the result will be $C = \text{Acc} \bmod F(x)$, where $F(x)$ is the field polynomial.

After each digit multiplication, the operand *A* has to be multiplied by x^D, therefore $A = A \times x^D \bmod F(x)$. To make the reduction by $F(x)$ possible in a single step, the following condition has to be fulfilled. If we represent a generating polynomial as $x^m+x^k+\ldots+1$, the reduction in a single step is possible when the digit size $D \leq m - k$. [3]

6. Proposed tests

Because our goal is to measure and compare the speed of the coprocessors implemented in a Combo6X card in the context of a whole PC system, we have to propose methods for precise measurement of the whole evaluation process.

Given a fixed frequency, we can compare the speed of the coprocessors by counting the number of clock cycles it takes to finish data processing. However, this is based solely on the design and the data processed and doesn't require a hardware implementation to be measured.

Our main focus therefore lies in measurement of the whole process, including the transfer of data over PCI. Each data transfer has two important parameters, throughput and latency [1]. With the maximal throughput 1.6Gbps, the transfer of data for a single key evaluation for m = 160 takes 500 ns, which doesn't cause any significant slow down of the coprocessor function. The main problem seems to be the latency.

Two problems with latency accompany the measurement. The first one is that the latency of the whole time measurement is comparable with execution time of a single key evaluation, which makes software measurement of just a single evaluation virtually impossible.

The obvious solution to this problem is to measure the time of several computations. There are two possible ways to do this. The first one is to simply perform a number of single evaluations. The second way implements a buffer on the card, transfers a batch of input data into the card, lets it all get processed and then fetches all the results, effectively using PCI burst mode in both directions.

However, both approaches have to deal with the second latency problem. This problem is caused by context switching. Let's assume we use batch processing of a thousand keys, using the hardware buffer variant. We assume that the whole evaluation will take about 40 ms. If the evaluation was done by software, we could simply measure CPU time of the given process. But, to measure the performance of hardware solution we need to perform the measurement using system time. If the measuring process loses con-

text close to the end of the processing and gains it back one time slot (i.e. usually 100 ms) later, the measured period will be almost 140 ms instead of expected 40 ms.

To solve this, we decided to divide the time measurement into three periods. During the first time period the transfer of input data is measured. During the second time period the processing of data inside the coprocessor is measured. Finally, in the last time period, the transfer of results from the coprocessor to host computer is measured. All three time periods will be measured in hardware by timers and the driving idea is that once the transfer of data is started, it won't be interrupted by a context switch. But if it happens, it can be easily detected as an extreme increase of the transfer time and the whole experiment will then be repeated. The final time will then be a simple sum of the three measured time periods, ignoring the possible delay between the end of processing and the start of the results transfer.

This method should give us good insight into how fast can the coprocessors perform in the context of a PC system, without the current system load interfering with the measurements.

7. Conclusions

We have described the basic principles and algorithms used for ECDSA and the current state of the coprocessor design. We proposed using a digit-serial multiplier for polynomial basis to gives us a better scalability of the whole processor, allowing us to maximize throughput of the processor at a given frequency.

We also described several key problems in the measurement of the performance of the whole system and proposed a method that will minimize influence of the system load on the measurement.

8. Acknowledgement

This work has been supported by CESNET, project 140R1/2005.

9. References

[1] Bečvář, M. - Schmidt, J.: Reconfigurable Acceleration of Intel PC: A Quantitative Analysis, Proceedings of IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop. Gyor: Széchenyi István University of Applied Sciences, 2001, s. 93-96. ISBN 963-7175-16-4.

[2] Borůvka, O.: Kryptografický procesor, CTU in Prague, Diploma thesis

[3] Guajardo, J. - Güneysu, T.- Kumar, S. S. - Paar, C. – Pelzl, J.: "Efficient Hardware Implementation of Finite Fields with Applications to Cryptography", Acta Applicandae Mathematicae: An International Survey Journal on Applying Mathematics and Mathematical Applications, Volume 93, Numbers 1-3, pp. 75-118, September 2006.

[4] IEEE P1363 Standard for Public-key Cryptography (Draft Version 13). IEEE, November 1999

[5] Itoh, T. - Teechai, O. – Tsujii, S.: A Fast Algorithm for Computing Multiplicative Inverse in GF(2t) using normal bases. J. Society for electronic Communications (Japan) 44 (1986), 31-36.

[6] Omura, J., Massey, J.: Computational Method and Apparatus for Finite Field Arithmetic. U.S. patent number 4,587,627, 1986