A Scalable Architecture for Elliptic Curve Point Multiplication

Kimmo Järvinen, Matti Tommiska and Jorma Skyttä
Helsinki University of Technology
Signal Processing Laboratory
Otakaari 5 A, FIN-02150, Finland
{kimmo.jarvinen}{matti.tommiska}{jorma.skytta}@hut.fi

Abstract

An architecture for elliptic curve point multiplication, which is developed especially for FPGAs, is introduced. The point multiplication is the basic operation of any elliptic curve cryptosystem and, hence, it must be implemented efficiently. The architecture is designed to be flexible in terms of speed and logic requirements and it can be scaled to meet the demands of different applications. Implementations have proven to be very efficient in performance and they belong to the fastest published FPGA-based ECC designs for most elliptic curve parameters.

1 Introduction

A scalable architecture for Elliptic Curve Point Multiplication (ECPM) is presented in this paper. ECPM is the basic operation of any elliptic curve cryptosystem and it is essential that it is implemented efficiently. Several implementations of ECPM have been introduced e.g. in [2], [5], [6], [9], [11] and [12]. Certain of these designs implement only the Galois field arithmetic part of an ECPM, but the architecture presented in this paper implements the entire ECPM allowing better performance, respectively.

ECC implementations can be categorized into reconfigurable and non-reconfigurable classes. In a reconfigurable implementation, the Galois field, over which the elliptic curve is defined, can be changed without the need to change the design. In a non-reconfigurable design, the FPGA must be reprogrammed in order to change the field. Non-reconfigurable designs are usually significantly faster than reconfigurable ones. Therefore, they should be preferred if very high performance is required. The architecture presented here is non-reconfigurable.

Reprogrammability of FPGAs is an advantage for non-reconfigurable implementations, because a non-reconfigurable architecture on the FPGA can be changed by reprogramming the device. Hence, reprogrammability vastly reduces disadvantages of non-reconfigurable architectures.

Field multiplication is the operation which ultimately defines the performance of an ECPM. Hence, it is essential that it is implemented carefully. The multiplier architecture presented here was designed on the lowest possible level to ensure maximum performance. Flexibility of the multiplier structure was the key issue in the design, because it is important that the implementation can be scaled to meet both speed and area requirements of an arbitrary application.

2 Elliptic Curve Cryptography

ECC has been of much interest in the cryptography community, because a high level of security can be achieved with short keys and low computational complexity. [3]

In this paper, only elliptic curves E over Galois field $GF(2^m)$ with a polynomial basis are considered. An irreducible polynomial generating $GF(2^m)$ is denoted as $m(x)$. ECPM is defined on E, so that

$$Q = kP = \underbrace{P + P + \ldots + P}_{k\text{ times}} \qquad (1)$$

where Q and P are distinct points on E and k is a large integer [3].

ECPM in Eq. (1) is calculated with successively performed operations called point addition and point doubling. A point addition is an operation $P_3 = P_1 + P_2$, where P_i are distinct points on an elliptic curve, and a point doubling is an operation $P_3 = 2P_1$.

Several efficient methods to compute ECPM in Eq. (1) have been proposed. A method presented by López and Dahab in [10] is used in the designs presented in this paper. The López-Dahab method comprises an ECPM algorithm and Madd, Mdouble and Mxy subroutines. The Madd algorithm is used for point additions and Mdouble for point doublings. Mxy is used for coordinate conversion.

3 Architecture for ECPM

The design presented in this paper implements an entire ECPM. In many publications, an ECC processor, implementing only the Galois field arithmetic

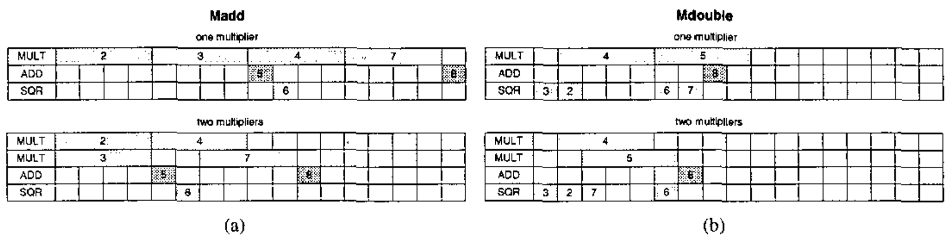

Figure 1. Schedules for the Madd (a) and Mdouble (b) algorithms with one and two multipliers

part of an ECPM, is designed. This approach leads to smaller designs, but better performance is achieved by implementing the entire ECPM. An important benefit is the processor off-load achieved with the complete implementation, i.e., the processor controlling the ECPM is freed completely from the process, while in ECC processor solutions the controlling processor has to control the use of the ECC processor.

If the ECPM method presented in [10] is used, the critical operation of ECPM is the Galois field multiplication. Thus, the multiplication must be implemented with special care. The multiplier architecture presented in this paper is non-reconfigurable.

The ECPM architecture was used in a VHDL generator called SIG-ECPM, where SIG is an acronym for the Signal Processing Laboratory at Helsinki University of Technology. SIG-ECPM is considered in more detail in [8] and this paper concentrates on the architecture of the SIG-ECPM designs.

The ECPM algorithm and its subroutines require that at least one multiplier, adder and inverter are available. Because a general multiplication requires considerably more time than a squaring, also a squarer is included. One inverter suffices, because there is only one inversion in an ECPM. It can be shown that a second adder or squarer do not accelerate ECPM calculation, if the latency of a multiplication is at least three [7]. However, adding a second multiplier into the design speeds up the calculation significantly. Let M be the number of multipliers, i.e. $M = 1, 2$. Schedules for Madd and Mdouble are presented in Fig. 1, where the operations are numbered as in [10]. Latencies of the subroutines decrease significantly if a second multiplier is used. Similar schedules can be derived also for Mxy.

A simplified block diagram of the ECPM architecture is presented in Fig. 2. Pmult (ECPM algorithm), Madd, Mdouble and Mxy blocks are implemented as FSMs (Finite State Machines) and they control the order in which their subroutines are performed. Pmult controls Madd, Mdouble, Mxy, adder and squarer. Madd, Mdouble and Mxy control the field arithmetic blocks. Performance of the ECPM design is ultimately defined by the field arithmetic blocks. They are discussed in Sec. 3.1 – 3.2.

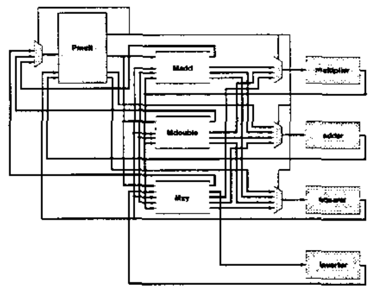

Figure 2. Block diagram of the proposed ECPM architecture

3.1 Multiplier Structure

Key objective in the derivation of the field multiplier architecture was that it can be scaled to meet both latency and area requirements. Thus, a scalable architecture was developed. The structure is developed particularly for FPGAs which commonly use 4-to-1-bit Look-Up Tables (LUTs) as their basic reprogrammable logic units, see e.g. [1] and [14]. A LUT is a block which can implement any 4-to-1-bit function. All optimizations are performed so that this structure is used as efficiently as possible. Although the architecture is developed for FPGAs, it can be used for ASIC implementations as well.

A field multiplication includes an algebraic multiplication of polynomials and a reduction modulo an irreducible polynomial $m(x)$. These phases can be calculated simultaneously as performed in many reconfigurable multipliers. If irreducibles are fixed, the reduction can be calculated in one clock cycle and it is beneficial to compute these phases separately.

The reduction is trivial if the irreducible is known *a priori*. Simple xor-equations can be derived using known algorithms given, e.g., in [3] and they can be hardwired into the design. The reduction can be calculated in one clock cycle with a high enough clock frequency, because irreducible polynomials used in ECC are usually trinomials or pentanomials.

The algebraic multiplication is calculated in structures called LUT-trees which consist of 4-to-1-bit

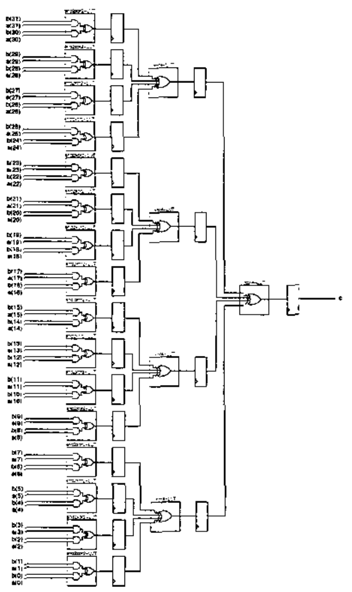

Figure 3. A 3-level LUT-tree

blocks, i.e., LUTs. An example of a LUT-tree is presented in Fig. 3, where the number of levels $\kappa = 3$, number of inputs $\tau = 32$, and number LUTs between registers $\lambda = 1$. The first level of a LUT-tree consists of and2xor2-LUTs. An and2xor2-LUT calculates $c = a_0b_0 + a_1b_1$. All other levels of the tree consist of xor4-LUTs. A xor4-LUT computes $c = a_0 + b_0 + a_1 + b_1$. Thus, a LUT-tree calculates

$$c = \sum_{i=0}^{\tau-1} a_i b_i \qquad (2)$$

where τ is the length of the inputs of the LUT-tree.

A coefficient of the result of an algebraic multiplication of two m-bit Galois field elements can be calculated through convolution as follows [6]

$$d_i = \sum_{k=0}^{i} a_k b_{i-k}. \qquad (3)$$

A LUT-tree calculates Eq. (3) if its inputs are set as follows: a-input is the element $a(x)$ and b-input is taken from a shift register which contains the coefficients of element $b(x)$ in reversed order. A LUT-tree calculates $d_0 = a_0b_0$ during the first clock cycle, $d_1 = a_0b_1 + a_1b_0$ during the second one, etc.

Eq. (3) sets a lower bound for the length of inputs τ. The length must satisfy $\tau \geq m$, because the maximum number of and-operations required in calculation of d_i is m, i.e. for coefficient d_{m-1}. However, smaller τ suffice for the rest of the coefficients.

In Fig. 3, there are registers between every LUT-level, but in real applications it is beneficial to compute several LUT-levels in a clock cycle. The best results were achieved if registers were added between every third level, i.e. $\lambda = 3$.

Let n be the number of LUT-trees and $\kappa_{\max}$ the number of levels in the largest LUT-tree. When calculation of coefficients d_i is divided equally for n LUT-trees, the latency of the multiplier is defined as

$$L_M = \text{ceil}\left(\frac{2m-1}{n}\right) + \text{ceil}\left(\frac{\kappa_{\max}}{\lambda}\right) + c \qquad (4)$$

where c is a constant defined by the design decisions, e.g., possible input/output registers. Eq. (4) can be used for scaling latency (L_M) and area (n) of the multiplier to fit the requirements of an application.

3.2 Other Galois Field Arithmetics

Implementation of the field addition is trivial as it is only an m-bit xor-operation. Field squaring is easy to perform if irreducible polynomials are fixed. The algebraic multiplication is performed only by adding zeros into the bit vector and no LUT-tree structure is needed. As mentioned in Sec. 3.1, reduction can be calculated with hardwired xor-equations derived using known algorithms. Field inversion is a more complex operation. The method used in the SIG-ECPM architecture was introduced by Shantz in [13].

4 Results

The ECPM architecture presented in Sec. 3 was implemented on Xilinx Virtex-II XC2V8000-5 which contains logic resources of 46,592 slices. A slice is the basic element of Xilinx FPGA devices and it consists of two LUTs, carry logic and two flip-flops [14].

Several implementations of the architecture were designed using SIG-ECPM VHDL generator [8] with different parameters. Elliptic curves, recommended by the Standards for Efficient Cryptography Group (SECG) in [4], were used in the implementations.

Results of the implementations are presented in Table 1. The latency value is calculated on an SECG

Table 1. Results of the SIG-ECPM implementations Virtex-II XC2V8000-5

m	M	L_M	Slices	Latency	Clock (MHz)	Time (μs)
113	2	8	10686 (22%)	6645	108.3	61
131	2	8	13264 (28%)	7667	93.2	82
163	2	8	18079 (38%)	9580	90.2	106
193	2	10	19250 (41%)	12535	90.2	139
233	2	12	23020 (49%)	16705	73.6	227
283	2	20	23736 (50%)	27026	75.5	358
409	1	20	32836 (70%)	62463	55.2	1132

curve [4] with one randomly selected integer k and it is given in clock cycles. It should be noticed that the latency values are not constants and also the ECPM times can be considered only as indicative results.

The results indicate that the architecture functions better, when smaller Galois fields are used. When field sizes grow, results degrade. The main reason for this are the area requirements which grow near to the limits of the target device causing slower results. If Galois fields are small, field multipliers with low latencies can be used with high clock frequencies, as can be seen in Table 1. When area requirements grow near to the limits of the device, place & route is a harder task than for smaller implementations.

5 Comparison

Certain FPGA-based ECC designs implemented on Xilinx devices are collected into Table 2 where the column R/N tells whether the design is reconfigurable (R) or non-reconfigurable (N). The SIG-ECPM results are the fastest in the comparison. For $m = 233$, the logic requirements become a constraint. Hence, poorer results are attained and Gura's design is slightly faster.

Table 2. FPGA-based implementations

Design	Device Family	m	R/N	Slices	Clock (MHz)	Time (μs)
SIG-ECPM	Virtex-II	113	N	10686	108.3	61
	"	163	N	18079	90.2	106
	"	193	N	19250	90.2	139
	"	233	N	23020	73.6	227
Bednara [2]	Virtex	191	R	n.a.	50	2270
Gura [5][6]	Virtex-E	163	N	n.a.	66.4	143
	"	193	N	n.a.	66.4	187
	"	233	N	n.a.	66.4	225
Eberle [5]	Virtex-II	163	R	n.a.	66.4	300
	"	193	R	n.a.	66.4	420
	"	233	R	n.a.	66.4	510
Nguyen [11]	Virtex-II	233	R	13180	n.a.	2979
Orlando [12]	Virtex-E	167	R	n.a.	76.7	210

The logic requirements of SIG-ECPM are higher than other reported requirements. However the design, where the logic requirements are given, is substantially slower than SIG-ECPM. Area requirements of the implementations, which have a similar level of performance, are not available. Anyhow, it can be assumed that SIG-ECPM requires more resources, because the entire ECPM is implemented. The comparison presented here can be considered only suggestive because different device families were used.

6 Conclusions

An architecture implementing an entire ECPM was introduced. Traditionally, published ECC designs implement only the Galois field operations but, by implementing the entire ECPM, faster performance was attained.

Galois field multiplication is the critical operation of an ECPM and it was designed with special care. The design was made on as low level as possible. Because the basic element of most modern FPGAs is a 4-to-1-bit LUT, the architecture was optimized to exploit this structure as efficiently as possible.

The architecture proved to be very efficient. For most parameters, it is the fastest published FPGA-based architecture for ECC at the time of writing this paper, at least to the authors' knowledge. The research was performed in the GO-SEC project at HUT.

7. References

[1] Altera. *Stratix II Device Handbook*, 2004.

[2] M. Bednara, M. Daldrup, J. von zur Gathen, J. Shokrollahi, and J. Teich. Reconfigurable Implementation of Elliptic Curve Crypto Algorithms. *Proc. of Int. Parallel and Distributed Processing Symposium, IPDPS'02, Fort Lauderdale, FL, USA*, pages 157–164, Apr. 15-19, 2002.

[3] I. Blake, G. Seroussi, and N. Smart. *Elliptic Curves in Cryptography, London Mathematical Society Lecture Note Series 265.* Cambridge University Press, 2002.

[4] Certicom Research. SEC 2: Recommended Elliptic Curve Domain Parametres. *Standards for Efficient Cryptography*, Sep. 20, 2000. Version 1.0.

[5] H. Eberle, N. Gura, and S. Chang-Shantz. A Cryptographic Processor for Arbitrary Elliptic Curves over $GF(2^m)$. *Proc. of Int. Conf. on Application-Specific Systems, Architectures, and Processors, ASAP'03, The Hague, The Netherlands*, pages 444–454, Jun. 24-26, 2003.

[6] N. Gura, H. Eberle, and S. C. Shantz. Generic Implementations of Elliptic Curve Cryptography using Partial Reduction. *Proc. of Conf. on Computer and Communications Security, CCS'02, Washington, DC, USA, ACM Press*, pages 108–116, Nov. 18-22, 2002.

[7] K. Järvinen. Hardware Description Language Generator for Elliptic Curve Point Multiplication. Master's thesis, Helsinki University of Technology, 2003.

[8] K. Järvinen, M. Tommiska, and J. Skyttä. A VHDL Generator for Elliptic Curve Cryptography. *Proc. of Int. Conf. on Field-Programmable Logic and Applications, FPL 2004, Antwerp, Belgium*, pages 1098–1100, Aug. 30-Sep. 1, 2004.

[9] T. Kerins, E. Popovici, W. Marnane, and P. Fitzpatrick. Fully Parametrizable Elliptic Curve Cryptography Processor over $GF(2^m)$. *Proc. of Int. Conf. on Field Programmable Logic and Applications, FPL 2002, Montpellier, France*, pages 750–759, Sep. 2-4, 2002.

[10] J. López and R. Dahab. Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation. *Proc. of Workshop on Cryptographic Hardware and Embedded Systems, CHES'99, Worcester, MA, USA*, pages 316–327, Aug. 1999.

[11] N. Nguyen, K. Gaj, D. Caliga, and T. El-Ghazawi. Implementation of Elliptic Curve Cryptosystems on a Reconfigurable Computer. *Proc. of Int. Conf. on Field-Programmable Technology, FPT 2003, Tokyo, Japan*, pages 60–67, Dec. 15-17, 2003.

[12] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. *Proc. of Workshop on Cryptographic Hardware and Embedded Systems, CHES'00, Worcester, MA, USA*, pages 41–56, Aug. 17-18, 2000.

[13] S. C. Shantz. From Euclid's GCD to Montgomery Multiplication to the Great Divide. Technical report, Sun Microsystems Laboratories, Jun. 2001.

[14] Xilinx, Inc. *Virtex-II Platform FPGAs: Complete Data Sheet*, Oct. 14, 2003.