TRADEOFF ANALYSIS OF FPGA BASED ELLIPTIC CURVE CRYPTOGRAPHY

M. Bednara, M. Daldrup, J. Teich

Computer Engineering Laboratory
University of Paderborn
Paderborn, Germany

*J. von zur Gathen, J. Shokrollahi**

Algorithmic Mathematics
University of Paderborn
Paderborn, Germany

ABSTRACT

FPGAs are an attractive platform for elliptic curve cryptography hardware. Since field multiplication is the most critical operation in elliptic curve cryptography, we have studied how efficient several field multipliers can be mapped to lookup table based FPGAs. Furthermore we have compared different curve coordinate representations with respect to the number of required field operations, and show how an elliptic curve coprocessor based on the Montgomery algorithm for curve multiplication can be implemented using our generic coprocessor architecture.

1. INTRODUCTION

Public key crypto systems are typically slower than symmetric systems but provide arbitrarily high levels of security and do not require an initial private key exchange. Due to the cost of the public key crypto algorithms, dedicated hardware support is desirable. FPGA cryptoprocessors avoid a series of drawbacks of ASIC systems:

- In electronic commerce servers, cryptography algorithms must be changed often for the purpose of adaption to the current workload, depending on the type of cryptography that is mainly used (public key or symmetric). This can be done by exploiting the FPGA's reconfiguration facility.

- FPGAs allow for an effortless adaption to changing security requirements.

In this paper, we present an area and performance tradeoff analysis for FPGA based implementations of a cryptography coprocessor using elliptic curve crypto algorithms. Among many others, FPGA based elliptic curve coprocessor implementations have been presented in [1], [2], [3], and [4]. [1] uses normal basis to represent the finite field and the Massey-Omura multiplier for field multiplication. The implementation in [2] uses the Montgomery method for point multiplication which is introduced in [5] and modified for fields of characteristic 2 in [6]. [3] is another implementation of the Montgomery method in the normal basis representation.

In this work we compare several ways for hardware implementation of elliptic curve point multiplication and focus on the effects that occur especially on FPGA based implementations. Our comparison includes different mixed coordinate representations for curve point coordinates which turned out to be more efficient than single coordinate representations. In addition we analyze the effect

---

of parallelism on the overall performance. This topic has been already briefly analyzed in [4]. This work is a continuation of [7], where some of the theoretical analyses were presented.

2. ELLIPTIC CURVE CRYPTOGRAPHY

Points of an elliptic curve defined over a finite field form a group, together with the point addition [8] as group operation. The basic operation in elliptic curve cryptosystems is the computation of $m\mathcal{P}$, where $\mathcal{P}$ is a curve point and m a large integer The computation of $m\mathcal{P}$ is a sequence of point additions and doublings. Elliptic curve cryptosystems (ECC) rely on the fact that solving the discrete logarithm problem on a elliptic curve is a hard task, that means, for a given $\mathcal{P}$ and m, computing $m\mathcal{P}$ is of polynomial complexity, but computing m by knowing only $\mathcal{P}$ and $m\mathcal{P}$ is of much higher complexity [9]. Our choice of 191-bit curves is considered secure at this time. The computation of $m\mathcal{P}$ is done in three abstraction levels:

- Algorithmic level (*double-and-add method, addition subtraction chains*)

- Curve arithmetic level (*Selection of coordinate representation*)

- Field arithmetic level (*basis selection, multiplier and inverter structures*)

Each level can be optimized in order to meet given area and performance constraints. Here, we focus on the curve and field arithmetic levels.

3. FINITE FIELD ARITHMETIC

Finite field operations constitute the base operations to elliptic curve computations, so optimizing them has a great impact on the multiplication performance. Here, we discuss the influence of the field multiplication on the overall latency of the $m\mathcal{P}$ multiplication.

3.1. Addition, Squaring and Inversion

Addition of two field elements of $GF(2^n)$ is simply a bitwise XOR combination of the corresponding bits of two n-bit vectors. The field basis representation has no effect on the performance of addition.

Squaring is a special case of multiplication but can be performed much more efficiently since both operands are equal. Squaring in the normal basis is a cyclic shift and in polynomial basis a shift followed by a reduction modulo an irreducible polynomial which

---

can be a trinomial or pentanomial (See [10]). Thus, squaring always is an inexpensive operation. ([10], [11]).

Inversion is the most complex field operation but can be performed as a sequence of multiplications and squarings using Fermat's little theorem. Since squaring operation is of low complexity, inversion is mainly affected by multiplication performance which is analyzed in the next section.

3.2. Multiplication

The most resource consuming base operation in finite fields is multiplication which can be affected by the base of representation. The most popular representations are polynomial and normal basis.

Finite field multipliers can be grouped into two major categories, namely parallel and serial multipliers. Parallel multipliers perform the total multiplication in one clock cycle but require large chip area. Examples of parallel multipliers are discussed in [12], [13], [14]. Serial multipliers have a less complex structure, but generate only a few result bits per clock cycle [15], [16], [17]. Serial multipliers are more attractive under hard area constraints, since they require less gates than parallel ones. In this section we study two popular multipliers, Massey-Omura for normal basis and LFSR for polynomial basis. Both can be implemented in a word-serial manner, i.e. a word (which is a fraction of a field element) of certain length is computed in parallel but multiple words are computed subsequently. This can be seen as a trade-off between the area-consuming parallel and time-consuming serial multipliers.

We compare the above mentioned multipliers with respect to three different technology models:

- **Theoretical analysis in 2-input gate model.** Most of the literature considers this model to analyze the hardware cost of algorithms. The model assumes only 2-input gates and neglects the influence of routing. The results are used for speed and area estimations of silicon implementations (e.g. ASICs).

- **4-input LUT[1] model.** FPGA use lookup table based logic elements that compute arbitrary boolean functions up to a certain number of inputs in constant time and area. We assume a 4-input gate model here since many popular FPGA (e.g. from Xilinx) use 4-input LUTs. Here we count only the number of LUTs required by a synthesized algorithm. The ratio of the number of 2-input gates to the number of 4-input LUTs is a number between 1 and 3, depending on the structure of the circuit. Routing is also neglected since it greatly depends on the selected FPGA routing resources. We compare the number of used LUTs of different multipliers.

- **Real FPGA implementations.** FPGA routing has a much higher influence on timing behavior than on ASIC implementations due to switching circuitry and buffer trees for high fanout nets. This model regards the experimental results of synthesis for a real FPGA under hard timing constraints. We have synthesized all multipliers for a Xilinx XCV2000E-FG1156-7 FPGA. [18]

The results of the comparison between the 2-input and the 4-input models are shown in Table 1. The latter completely agree with a theoretical 4-input analysis for the number of elements.

---
[1]Lookup table.

Word length	2-input gate		4-input LUT			
	LFSR	M-O	LFSR	r	M-O	r
1	384	761	380	1.01	253	3.01
2	773	1522	384	2.01	506	3.01
4	1549	3044	774	2.00	1012	3.01
8	3101	6088	1168	2.65	2024	3.01
16	6205	12176	2652	2.34	4048	3.01
32	12413	24352	4820	2.58	8096	3.01
50	20189	38050	7191	2.81	12650	3.01
64	24829	48704	8953	2.77	16192	3.01
100	40389	76100	13684	2.95	25300	3.01

Table 1. Number of logic elements required for LFSR and Massey-Omura (M-O) multipliers for $GF(2^{191})$ in the 2-input gate model (given by a theoretical analysis) and the 4-input LUT model (by experimental results). The column r shows the ratio of 2-input-gates to 4-input-LUTs.

In the best case, every 4-input LUT can substitute three 2-input gates. As it can be seen from Table 1 Massey-Omura makes the best use of this reduction in the number of LUTs, and that the LFSR multiplier only approaches a reduction factor of 3 for larger word lengths. For the same reason, there is almost no difference between the number of LUTs when implementing LFSR multipliers with the word lengths of 1 and 2 respectively. LFSR, however, requires a significantly lower number of LUTs.
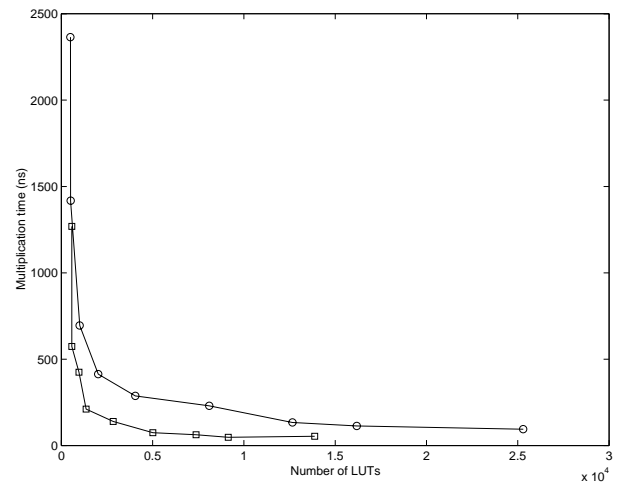

Fig. 1. Multiplication time versus number of LUTs for LFSR and Massey-Omura multipliers in $GF(2^{191})$ (Lower and upper curves respectively).

Figure 1 shows the time required for a complete multiplication for a given number of LUTs in $GF(2^{191})$. As can be seen, LFSR multiplier is always faster than the Massey-Omura multiplier for a fixed number of LUTs. The LFSR multiplier seems to be the better choice but the following facts should be taken into account:

- LFSR implementations cause an irregular FPGA floorplan that may result in poor timing behavior depending on the FPGA type.

- Massey-Omura multipliers are used for normal basis rep-

resentations where the squaring operation is faster than in the polynomial base since no final modulo reduction is required.

4. POINT ADDITION AND DOUBLING

The task of point multiplication is the computation of $m\mathcal{P}$ based on repeated point addition and doubling operations. In this section we consider different point coordinate representations and study their effect on the point addition and doubling costs.

4.1. Coordinate Representations

Most point multiplication methods are based on repeated addition/subtraction and doubling of points (see Section 4.2). In this subsection, we compare different representations of points on elliptic curves and show how hybrid forms of them can enhance the computation logic depending on various area and performance constraints. Hybrid coordinate representations were originally discussed in [19], see also [20].
Coordinate representations considered here are the Affine, Projective, Jacobian and López-Dahab representations [20]. Transforming affine coordinates into one of the other representations is almost trivial, but not vice versa since field inversion is required. Table 2 gives the formulas for mapping into affine coordinates.

Representation	Mapping to Affine Coordinates
Projective	$x = X/Z, \; y = Y/Z$
Jacobian	$x = X/Z^2 \; y = Y/Z^3$
López-Dahab	$x = X/Z \; y = Y/Z^2$

Table 2. Mapping of Projective, Jacobian, and López-Dahab representation into affine coordinates.

We have analyzed Affine/Jacobian and Affine/López-Dahab hybrid coordinate representations, Moreover, we analyzed the Montgomery method [6] which is actually not a hybrid coordinate representation but is based on the Jacobian coordinates.

Table 3 compares the methods for point addition and doubling in these mixed coordinate representations with respect to the number of multiplications $\mathcal{M}$, squarings $\mathcal{S}$ and additions $\mathcal{A}$ in the underlying finite field.

Representation	Addition	Doubling
Affine/López-Dahab	$10\mathcal{M} + 5\mathcal{S} + 8\mathcal{A}$	$5\mathcal{M} + 5\mathcal{S} + 4\mathcal{A}$
Affine/Jacobian	$14\mathcal{M} + 4\mathcal{S} + 10\mathcal{A}$	$6\mathcal{M} + 4\mathcal{S} + 4\mathcal{A}$
Montgomery	$4\mathcal{M} + 1\mathcal{S} + 2\mathcal{A}$	$2\mathcal{M} + 4\mathcal{S} + 1\mathcal{A}$

Table 3. Costs of different point addition and doubling methods in terms of field operations.

4.2. Point Multiplication Algorithms

The best known point multiplication algorithm is the double and add method which uses the binary representation of m.

An arbitary number m satisfying $2^{n-1} < m < 2^n$ has an average of $\frac{n}{2}$ number of 1-bits. The number of operations can be reduced by using addition-subtraction chains [21] requiring only $\frac{n}{3}$ additions while the number of doublings is fixed.
The selection of the point multiplication method belongs to the algorithmic abstraction level (see Section 2) and is not considered here.

5. IMPLEMENTATION OF THE MONTGOMERY ALGORITHM

In [7] we have presented a generic datapath architecture that can be easily configured to meet various performance/area constraints by exploiting several degrees of freedom: the number of functional units in the data path, the degree of parallelism in the multipliers, the type of point multiplication algorithm, the type of the controller and the coordinate representation. For performance measures, we have implemented a sample architecture on a prototyping system with a Xilinx Virtex-1000 FPGA (XCV1000-BG560-4).
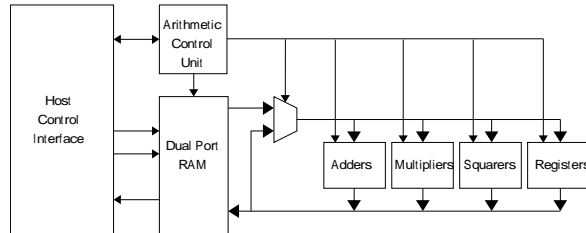

Fig. 2. Datapath structure.

The structure of the datapath architecture is shown in Fig. 2. We have used two squarers, two adders and four sequential LFSR multipliers in our sample architecture presented in [7]. Here, we use a different configuration of the processor architecture which is optimized to compute a curve multiplication using the Montgomery algorithm.

5.1. Montgomery Algorithm

We give a short introduction into the idea of the Montgomery multiplication of elliptic curve points. The algorithm is outlined in Fig. 3.

Set $m \leftarrow (m_{l-1}m_{l-2}\ldots m_1 k_0)_2$.
Set $P_1 \leftarrow P, P_2 \leftarrow 2P$.
for i **from** $l-2$ **downto** 0 **do**
　if $m_i = 1$ **then**
　　Set $P_1 \leftarrow P_1 + P_2, P_2 \leftarrow 2P_2$
　else
　　Set $P_2 \leftarrow P_1 + P_2, P_1 \leftarrow 2P_1$
　end if
end for

Fig. 3. The Montgomery algorithm for elliptic curve point multiplication

This algorithm computes the product $m\mathcal{P}$ where m is a large integer and $\mathcal{P}$ an elliptic curve point.
The algorithm processes sequentially the bits of m. In each iteration, a point addition and a point doubling is performed. The parameters of these operations depend on the state of the currently
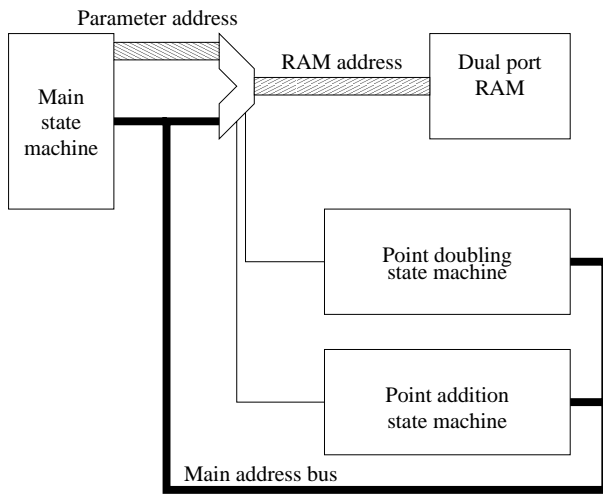
Fig. 4. Address bus structure for parameter passing.

processed bit of m.

The coordinates of the curve points P_1 and P_2 are stored in fixed locations in the dual ported operand memory (refer to [7]). In order to avoid additional overhead for writing parameters into temporary registers and back to memory after the computation, we have used a special address logic structure which allows for indirect addressing. This structure is shown in Fig. 4. The main state machine puts the address of the apropriate variable on its parameter address port before triggering the sub-state machines for doubling or addition. Only one state machine (for point doubling or addition) is active at one time. Via the address multiplexer, the active state machine can select if the parameter address is passed on the DPRAM's address bus or (e.g. for temporary variable access) another address generated in the sub-state machine itself.

A maximum number of two field multiplications is performed *in parallel* by the Montgomery algorithm except for the last iteration, where more than two multiplications can be perfomed in parallel. Thus we use only two multipliers instead of 4 as in [7]. We have used word-parallel LFSR multipliers with a word length of 50 bits. The complete design uses about 17000 4-input LUTs, 3600 flipflops and can operate at 20 MHz on a XCV2000E-FG1156-7 FPGA. Each field multiplication requires 4 clock cycles.

6. SUMMARY

We have provided a tradeoff analysis of word-serial Massey-Omura and LFSR multipliers with respect to area and performance for FPGA implementations. Both multipliers were analyzed in a classical 2-input gate model and in a 4-input LUT model for FPGA. For a comparison of the theoretical results with real synthesis results, we have synthesized the multipliers for different word lenghts on a Xilinx Virtex FPGA. We could show that the synthesis results largely match the theoretical results. As a case study, we have implemented a elliptic curve coprocessor based on the Montgomery algorithm using our generic architecture model.

7. REFERENCES

[1] Lijun Gao, Sarvesh Shrivastava, and Gerald E. Sobelman, "Elliptic Curve Scalar Multiplier Design Using FPGAs," in *CHES '99*, 1999, number 1717 in LNCS, pp. 257–268.

[2] G. Orlando and C. Paar, "A High-Performance Reconfigurable Elliptic Curve Coprocessor for $GF(2^m)$," in *CHES '2000*, 2000, number 1965 in LNCS, pp. 41–56.

[3] Markus Ernst and Sorin Huss, "Ein FPGA basierter Elliptic-Curve Kryptoprozessor mit variabler Schlüssellänge für hohen Datendurchsatz," in *Entwurf Integrierter Schaltungen und Systemen*, Dresden, April 2001.

[4] N. P. Smart, "The Hessian form of an elliptic curve," in *CHES '2001*, 2001, number 2162 in LNCS, pp. 118–125.

[5] Peter L. Montgomery, "Speeding the Pollard and Elliptic Curve Methods of Factorization," *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, January 1987.

[6] Julio López and Ricardo Dahab, "Fast Multiplication on Elliptic Curves over $GF(2^m)$ without Precomputation," in *CHES '99*, 1999, number 1717 in LNCS, pp. 316–327.

[7] Marcus Bednara, Michael Daldrup, Joachim von zur Gathen, Jürgen Teich, and Jamshid Shokrollahi, "Reconfigurable Implementation of Elliptic Curve Crypto Algorithms," in *9th Reconfigurable Architecture Workshop (RAW 2002) to appear.*, 2002.

[8] Ian Blake, Gadiel Seroussi, and Nigel Smart, *Elliptic Curves in Cryptography*, Number 265 in London Mathematical Society Lecture Note Series. Cambridge University Press, 1999.

[9] Alfred J. Menezes, Ian F. Blake, XuHong Gao, Ronald C. Mullin, Scott A. Vanstone, and Tomik Yaghoobian, *Applications of finite fields*, Kluwer Academic Publishers, 1993.

[10] Joachim von zur Gathen and Michael Nöcker, "Exponentiation using addition chains for finite fields," submitted, 2002.

[11] Huapeng Wu, "On Computation of Polynomial Modular Reduction," Tech. Rep., Centre of Applied Cryptographic Research, University of Waterloo, June 2000.

[12] B. Sunar and Ç. K. Koç, "Mastrovito Multiplier for All Trinomials," *IEEE Transactions on Computers*, vol. 48, no. 5, pp. 522–527, July 1999.

[13] A. Halbutogullari and Ç. K. Koç, "Mastrovito Multiplier for General Irreducible polynomials," *IEEE Transactions on Computers*, vol. 49, no. 5, pp. 503–518, May 2000.

[14] Christof Paar, "A New Architecture for a Parallel Finite Field Multiplier with Low Complexity on Composite Fields," *IEEE Transactions on Computers*, vol. 45, no. 7, pp. 856–861, July 1996.

[15] G. Orlando and C. Paar, "A Super-Serial Galois Fields Multiplier for FPGAs and its Application to Public-Key Algorithms," in *FCCM '99*, April 1999.

[16] C. C. Wang, T. K. Truong, H. M. Shao, L. J. Deutsch, J. K. Omura, and I. S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Transactions on Computers*, vol. C-34, pp. 709–717, 1985.

[17] Çetin K. Koç and Tolga Acar, "Montgomery Multiplication in $GF(2^k)$," *Designs, Codes and Cryptography*, vol. 14, no. 1, pp. 57–69, April 1998.

[18] Xilinx, "Virtex-E 1.8V FPGAs, Detailed Description, http://www.xilinx.com/partinfo/ds022-2.pdf," Tech. Rep., Xilinx, Inc., 2001.

[19] Henri Cohen, Atsuko Miyaji, and Takatoshi Ono, "Efficient elliptic curve exponentiation using mixed coordinates," in *ASIACRYPT 1998*, 1998, number 1514 in LNCS, pp. 51–65.

[20] Julio López and Ricardo Dahab, "Improved Algorithms for Elliptic Curve Arithmetic in $GF(2^n)$," in *Selected Areas in Cryptography*, number 1556 in LNCS, pp. 201–212. 1998.

[21] Daniel M. Gordon, "A survey of Fast Exponentiation Methods," *Journal of Algorithms*, vol. 27, pp. 129–146, 1998.