Performance of HECC Coprocessors Using Inversionfree Formulae

Thomas Wollinger[1], Guido Bertoni[2], Luca Breveglieri[3] and Christof Paar[1]

[1] Communication Security Group (COSY)
Ruhr-Universitaet Bochum, Germany
{wollinger,cpaar}@crypto.rub.de
[2] STMicroelectronics,
Advanced System Technology Agrate Brianza - Milano, Italy,
guido.bertoni@st.com,
[3] Politecnico di Milano, Italy
{breveglieri}@elet.polimi.it

Abstract. The HyperElliptic Curve Cryptosystem (HECC) was quite extensively studied during the recent years. In the open literature one can find results on improving the group operations of HECC as well as implementations on various types of processors. There have also been some efforts to implement HECC on hardware devices, like for instance FPGAs. Only one of these works, however, deals with the inversionfree formulae to compute the group operations of HECC.
We present inversionfree group operations for the HEC $y^2 + xy = x^5 + f_1x + f_0$ and targeting characteristic two fields. The reason being to allow a fair comparison to hardware architectures using the affine case presented in [BBWP04]. In the main part of the paper we use these results to investigate various hardware architectures for a HECC VLSI coprocessor. If area constraints are not considered, scalar multiplication can be performed in 19769 clock cycles using three field multipliers (of type $D = 32$), one field adder and one field squarer, where D indicates the digit size of the multiplier. However, the optimal solution in terms of latency and area uses two multipliers (of type $D = 4$), one addition and one squaring. The main finding of the present contribution is that coprocessors based on the inversionfree formulae should be preferred compared to those using group operations containing inversion. This holds despite the fact that one field inversion in the affine HECC group operation is traded by up to 24 field multiplications in the inversionfree case.

Keywords: hyperelliptic curve cryptosystem, inversionfree formulae, hardware architecture, high performance explicit formulae, VLSI coprocessor.

1 Introduction

Koblitz and Miller proposed, independently from each other, Elliptic Curves (EC) for public-key cryptography. The generalizations of ECs are

called HyperElliptic Curves (HEC) and were first proposed for cryptographical use in [Kob88]. It is important to point out that hyperelliptic curve cryptosystems are promising because of the *short* operand sizes compared to other public key schemes.

We introduce an analysis of the hardware implementation of the inversionfree HEC group operations, including a comparison with affine hardware architectures. Previously there have been some efforts for implementing a HECC coprocessor on FPGAs; however our contribution is the first that contains a thorough analysis of the different design options and a comparison between the two possible coordinate systems (affine and projective). Our analysis is based on the derived group operations and the underlying field $\mathbb{F}_{2^{83}}$.

We investigated various hardware architectures by parallelizing the scalar multiplication of HECC at the following three levels: the field operation level, the group operation level and the scalar multiplication level. At the field operation level we used multipliers that handle different numbers of bits in parallel. At the group operation level we used different numbers of field multipliers. Our investigation of the parallelism reachable for the scalar multiplication level was achieved by overlapping the computation of consecutive group operations.

The scalar multiplication can be performed most efficiently in 19769 clock cycles using the coprocessor design providing three field multipliers (of type $D = 32$), one field adder and one field squarer. In order to find the optimal design for the HECC coprocessor, we considered speed as well as (silicon) area, namely the area-time product. The lowest area-time product was achieved by using two multipliers (of type $D = 4$), one adder and one squarer.

Furthermore, we compare the results of this paper with those published for the group operations using inversion [BBWP04]. Thus we found out that the most efficient way to implement a HECC VLSI coprocessor is to use inversionfree formulae. If we compare the best coprocessor configuration for affine coordinates and that using inversionfree formulae, we see that a) the latency is up to a factor of almost 3 better and b) the area-time product figure for the inversionfree cases is always better. Hence, HECC coprocessors should be based on the inversionfree coordinate system. This result could be achieved even with trading one inversion in the affine group operation with up to 24 field multiplication in the inversionfree case.

The paper is organized as follows. Section 2 presents our improved inversionfree HEC group operations and Section 3 describes the architec-

ture of the HECC coprocessor as well as the different levels of parallelization. In Section 4 we present and discuss results. Conclusions list final considerations.

2 Inversionfree Formulae

In the recent years a considerable effort has been focused on improving the group operations of the hyperelliptic curve cryptosystem (HECC), see [Wol04] for a summary. Analogously to EC, there also exist inversionfree group operations for HECC [MDM$^+$02, Lan03]. Not having to use inversions might also be profitable for HECC, however the extra cost in multiplications is very high compared to the simpler ECC case. We investigated the needed field operations of the inversionfree group operations considering a genus-2 HEC $y^2 + h(x)y = f(x)$, with $h(x) = x$, and $f(x) = x^5 + f_1x + f_0$. In addition, the characteristic of the underlying field was fixed to characteristic two. We do not know of any security limitations using this kind of curves. The main reason of using this parameters, was to provide a fair comparison to the affine HECC coprocessor presented in [BBWP04]. Furthermore, we were able reduced the number of finite field operations, see Table 1.

Table 1 lists our results with the best ones previously published and sets them in perspective with the affine group operations used in [BBWP04].

Table 1. Efficient inversionfree group operations for genus-2 HEC.

coordinate system		curve properties	addition	doubling
affine	[Lan03]	$h_2 = 0, h_i \in \mathbb{F}_2$ $f_4 = 0$	I+21M+3S	I+17M+5S
	[PWP04]	$h(x) = x$ $f_4 = f_3 = f_2 = 0$	-	I+9M+6S
projective	[MDM$^+$02]	N.A.	67M	42M
	[Lan03]	$h_2 = 0, h_i \in \mathbb{F}_2$ $f_4 = 0$	47M + 4S	40M + 7S
	our work	$h(x) = x$ $f_4 = f_3 = f_2 = 0$	45M + 5S	31M + 6S

3 Architecture of the HECC Coprocessor

Our coprocessor is designed according to a rather standard architecture consisting of a 3-bus loop scheme connecting a set of functional units

with a register file. A control unit drives the various function units and the register file. The register file stores temporary intermediate values and final results. The size of each register is the dimension of the field, namely 83 bits. The register file has two output ports to feed the operators to the function units and one input port to receive the result. The processor allows to load two elements and store one element at any clock cycle. This guarantees feasibility and ease of implementation. However, at any given clock cycle only one field operation can start. If the operation is unary, such as inversion, one input bus remains idle.

In Table 2 we give the area and latency for each arithmetic function unit we used. The given estimates assume 2-input gates and optimal field generator polynomials $F(x) = x^m + \sum_{i=0}^{t} f_i x^i$, where $m - t \geq D$. One observes that the gate-consuming functional units are multipliers and inverter, while in comparison the area used for the adder and squarer is negligible. In the case of multipliers and inverter the number of XOR gates and AND gates are the same. This two considerations allow us to equal the cost of a XOR gate and of an AND gate, since we will not use absolute value but relative value when comparing different system configuration.

We have considered that the different components of the system work at the same frequency. This might not be true. Usually, the frequency of this type of cryptosystem is imposed by the multiplier, and smaller digit-size will yield to higher clock frequency and thus speed-up these systems. For a correct estimation of the impact of the different clock frequencies a synthesis is necessary.

Table 2. Components of the coprocessor: area and time.

	Area		Latency
		Gate count	[clock cycles]
Add	$[m]$ XOR	$[m]$	1
Sqr [OP00]	$[4(m-1)]$ XOR	$[4(m-1)]$	1
Mul [SP97]	$[D \cdot m]$ AND & $[D \cdot m]$ XOR	$[2Dm]$	$\lceil m/D \rceil$
Inv [BCH93]	$[6 \cdot m + log_2 m]$ AND & $[6 \cdot m + log_2 m]$ XOR	$[2(6m + log_2 m)]$	$2 \cdot m$

In our analysis we considered different levels of parallelization for the HECC coprocessor:

1. Parallelization at field operation level: we achieved it by varying the digit size of the multiplier.
2. Parallelization at group operation level: our architecture of the HEC coprocessor allows to change the number of used multipliers.
3. Parallelization at scalar multiplication level: we achieved it by overlapping two consecutive group operations.

Taking into consideration all the different parallelization levels, we were able to identify the best architecture considering the execution time as well as the area requirements.

We coded a software library that schedules the HECC scalar multiplication mapping it onto the proposed hardware architecture. This software tool applies the so-called Operation Scheduling [Gov03] procedure for parallelizing the sequence of group operations computing the HECC scalar multiplication, and it works according to the As Soon As Possible (ASAP) scheduling policy. We then constrain the scheduling policy by imposing limits on the available hardware resources (i.e. type and number of arithmetic function units) and realistic time delays required to execute the field operations. A more detailed description of the scheduling methodology can be found in [BBWP04].

4 Analysis and Results

The main contribution of this paper is to identify the optimal architecture for the HECC coprocessor. In order to do so we first need to investigate the different architectures for the inversionfree group operations. Afterwards our results are compared to those valid for affine coordinates, as they are presented in [BBWP04].

We target genus-2 HECs defined over a finite field $\mathbb{F}_{2^{83}}$ using the derived group operations represented by using projective coordinates. As for the evaluation of the latency of the scalar multiplication operation (the so-called kD operation), we examined some average cases. Hence the 160-bit long integer k contains the same number of 0s and 1s.

All our results are presented against different digit sizes and varying numbers of multiplier units, representing the parallelism at the bit level and the field operation level, respectively. The parallelism on the scalar multiplication level, namely the overlapping between the group operations, is applied in the numbers given for the scalar multiplication and the area-time product. We choose not to upper bound the number of used registers. All the considered system configurations require 21 registers for storing temporary values, where each register stores a field element of

83 bits. One could reduce the number of registers at the cost of some additional latency by avoiding the overlapping of two consecutive group operations, for example.

4.1 Latency

In this subsection we present our results targeting the latency of the complete scalar multiplication, see Table 3.

Table 3. Latency of the scalar multiplication (in clock cycles, group order $\approx 2^{160}$).

Digit-size	Number of multipliers			
	1	2	3	4
2	356537	193176	130652	98888
4	181670	97355	69730	55646
8	98400	53243	38935	31953
16	56525	32835	24747	22974
32	31702	21209	19769	20490

The conclusion one can draw from the table is that with increasing hardware resources (more multiplier units and higher digit size), latency drops. Thus we can compute the scalar multiplication of HECC in a shorter time. Scalar multiplication can be performed most efficiently in 19769 clock cycles providing three field multipliers (of type $D = 32$), one field adder and one field squarer (Table 3, bottom row).

Our scheduler is based on the method known as Operation Scheduling (and works according the ASAP scheduling policy), which does not grant an optimal solution [Gov03]; this is evident in the case of 4 multipliers of type $D = 32$ (Table 3, 5-th row).

When carefully inspecting the results, we see that adding extra hardware resources might be unnecessary.

4.2 Area-Time Product

The optimal implementation will achieve the highest throughput consuming the smallest area (contrary to some traditional cryptographic implementations where only best performances were taken into care). Hence we consider both the hardware requirements and the time constraints of the cryptographic application.

In Table 4 we show the area-time product for the different design options. The analysis uses the normalized area-time product with respect

Table 4. Area-time product for scalar multiplication.

Digit-size	Number of multipliers			
	1	2	3	4
2	1,2207	1,1024	1,0438	1,0157
4	1,0367	1	1,0346	1,0796
8	1,0107	1,0330	1,1109	1,2034
16	1,0967	1,2367	1,3839	1,7043
32	1,1940	1,5733	2,1885	3,0167

to the lowest area-time product. Table 4 shows that the architecture using two multipliers (of type $D = 4$), one adder and one squarer achieves the best area-time product and therefore is the optimal architecture.

4.3 Comparing Affine and Projective HECC

In this section we put our results into perspective with the HECC VLSI coprocessor implementing affine coordinates, that is presented in [BBWP04][4]. This comparison is possible, because of the same methodology used, e.g. using the same curve parameter. Figure 1 compares the latency of affine

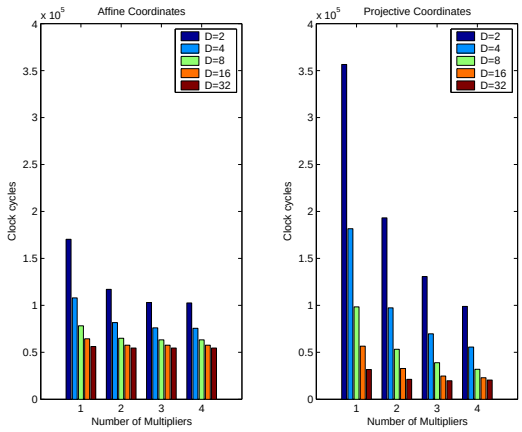

Fig. 1. Latency comparison between affine and projective HECC coprocessors.

and projective HECC coprocessors. Hence the lower bars are preferable. One can draw the following conclusions:

[4] Note, that it is not fair to compare our results with the previous work implementing HECC on FPGAs. The reason being the different architectures used.

- In both coordinate systems latency drops by providing additional hardware resources (increasing the digit size and the number of multiplier units)
- If the context allows to use projective coordinates, they are always preferable in terms of latency and area. However, affine coordinates might find an appropriate use in low area applications.
- One very important result in this contribution is that high speed implementations should definitely use projective coordinates. As it can be seen in Figure 1, projective coordinates using large digit sizes result in the lowest latency.

If comparison between affine and projective coordinates is limited only to scalar multiplication latency, then projective processor is better than affine one. The reason being the design based on projective coordinates can calculate a scalar multiplication in 19769 clock cycles, while affine can not be faster than 54593 clock cycles. This means a speed-up of 2.7.

In Figure 2 we compare the five best Area-Time (AT) product figures using the two different coordinate systems. For each system configuration the area-time product and the latency is reported. These both measures are normalized using the values of the system configuration with the best AT product. The left-most bars show the figures for affine coordinates and the five right-most bars show those for the projective case. The design option used for the HECC VLSI coprocessor is given under each bar (M denotes the multipliers).

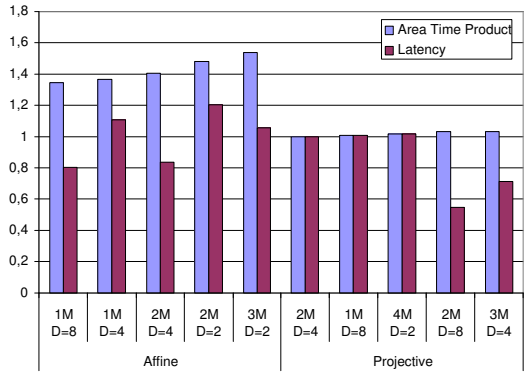

Fig. 2. Comparison of the best five area-time product figures using and an affine and a projective HECC coprocessor.

It is interesting to analyze the implicit parallelism using projective formulae. Analyzing the best five area-time products there is only one system configuration with one multiplier. In the case of affine coordinates we see that the two best solutions contain only one multipliers. Hence, potentially we cannot parallelize as many field operations.

Another general statement one can draw form Figure 2 is that systems using projective coordinates are always better in terms of the area-time product. We realize that there is a gap of about 35% in the area-time product between affine and projective coordinates.

Note that projective coordinates are not always better in term of latency. The reason being the area-time product metric. Hence, the projective system reaching a lower latency and a better AT product compared to affine have smaller area requirements. However, if we look at systems with similar area usage requirements, e.g., comparing the projective system using 3M and D=4 and the affine with 2M and D=4. In this case the AT product as well as the latency and the area of coprocessor based on projective coordinates are better. Latency is 17% better than affine.

Hence, the *main* finding of this contribution is that *projective* coprocessors are more flexible than affine ones, allowing the designer to chose the best compromise in terms of required latency and silicon area. If the application imposes to perform data conversion as well, affine coprocessors become attractive for low area requirements.

5 Conclusions

We presented newly derived explicit formulae for HECC using the inversionfree approach. Our formulae are up to 22.5% better than the best previous one and were the basis of an extensive study of different architecture options for a HECC VLSI coprocessor. We analyzed the parallelization at field operation level, group operation level and scalar multiplication level. The analysis was done using a HEC $y^2 + h(x)y = f(x)$, with $h(x) = x$, and $f(x) = x^5 + f_1x + f_0$ and a base field $\mathbb{F}_{2^{83}}$.

Comparing our finding for projective coordinates with that using affine coordinates for HECC, we can draw the following conclusions: a) considering only the latency the coprocessors based on projective coordinates lead to a higher performance (this system is up to a factor of almost 3 better), and b) considering the area-time product the projective coprocessor is preferable as well. Hence, the group operations computed using inversionfree formulae should be the appropriate choice of any HECC VLSI hardware implementation.

References

[BBWP04] G. Bertoni, L. Breveglieri, T. Wollinger, and C. Paar. Finding Optimum Parallel Coprocessor Design for Genus 2 Hyperelliptic Curve Cryptosystems. In *International Conference on Information Technology: Coding and Computing - ITCC 2004*. IEEE Computer Society, April 2004.

[BCH93] H. Brunner, A. Curiger, and M. Hofstetter. On Computing Multiplicative Inverses in GF(2^m). *IEEE Transactions on Computers*, 42:1010–1015, August 1993.

[Gov03] R. Govindaraian. *Instruction Scheduling*. CRC Press, The Compiler Design Handbook edition, 2003. editor Y. N. Srikant and P. Shankar.

[Kob88] N. Koblitz. A Family of Jacobians Suitable for Discrete Log Cryptosystems. In Shafi Goldwasser, editor, *Advances in Cryptology - Crypto '88*, LNCS 403, pages 94 – 99, Berlin, 1988. Springer-Verlag.

[Lan03] T. Lange. Formulae for Arithmetic on Genus 2 Hyperelliptic Curves. to appear in J. AAECC, September 2003. `http://www.ruhr-uni-bochum.de/itsc/tanja/preprints/expl_sub.pdf`.

[MDM+02] Y. Miyamoto, H. Doi, K. Matsuo, J. Chao, and S. Tsuji. A Fast Addition Algorithm of Genus Two Hyperelliptic Curve. In *The 2002 Symposium on Cryptography and Information Security — SCIS 2002, IEICE Japan*, pages 497 – 502, 2002. in Japanese.

[OP00] G. Orlando and C. Paar. A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$. In Ç. K. Koç and C. Paar, editors, *Cryptographic Hardware and Embedded Systems — CHES 2000*, LNCS 1965, pages 41 –56. Springer-Verlag, 2000.

[PWP04] J. Pelzl, T. Wollinger, and C. Paar. High Performance Arithmetic for Special Hyperelliptic Curve Cryptosystems of Genus Two. In *International Conference on Information Technology: Coding and Computing - ITCC 2004*. IEEE Computer Society, April 2004.

[SP97] L. Song and K. K. Parhi. Low-Energy Digit-Serial/Parallel Finite Field Multipliers. *Journal of VLSI Signal Processing Systems*, 2(22):1–17, 1997.

[Wol04] T. Wollinger. *Software and Hardware Implementation of Hyperelliptic Curve Cryptosystems*. Europäischer Universitätsverlag, 2004.