# High Speed Elliptic Curve Cryptography Processor for $GF(p)$

Christian Pühringer cip@gmx.at
Matr. Nr.:xxxxxx

8.7.2005

**Abstract**

This paper presents an elliptic curve cryptography processor for prime fields implemented on a FPGA-PCI board. It serves as a coprocessor to acclerate cryptographic operations, like signature generation and verification. It is intented to be used on systems with a high load of such operations, like webservers for e-government applications. Performance estimates based on first synthesis results suggest that more than 10000 point multiplications over a 192 bit prime field per second can be performed on a high end FPGA, which can easily compete with the fastest reported ASIC implementations. Targeted on low-end to mid-range FPGAs it is still the fastest FPGA implementation reported for elliptic curve cryptography over prime fields.

## 1 Introduction

Internet applications with high security requirements gain more and more importance. For applications, like e-Government or online banking, security and the trust of the potential users in their security is an important factor for their success. To achieve this high level of security encryption and authentication is used. There are several algorithms which can be used, but all share the property that they require large amounts of computing power. As many potential devices on the client side, like smart cards or mobile phones, cannot supply this computing power easily many hardware implementations of cryptographic algorithms where proposed and are in use. On the other hand server systems do have enough computing power to handle the cryptographic algorithms, therefore much less hardware acceleration solutions exist. However, while the client devices normally only have to calculate a single cryptographic operation, for example to calculate the digital signature for a document, the server systems have to handle many cryptographic operations in every second. For example, an e-Government server may have to verify signatures of thousands of digitally submitted documents every second. Therefore a high load is generated on the system and it clearly makes sense to use a cryptographic hardware acceleration device, which can process the cryptographic algorithms much more efficiently than a general purpose processor. Thus a smaller server can be used to handle the same amount of users and so the total costs of the system could be lowered significantly.

For the widely used *RSA* encryption system some server side solutions, like presented in Wöckinger [2005] exist. However, RSA has the disadvantage that it requires large keys, at least 1024 bit, to achieve a basic level of security. This requires large and expensive hardware both at the client and at the server side. *Elliptic curve cryptography (ECC)* is a very good alternative, as it allows much smaller key sizes, and therefore much cheaper hardware, than RSA. According to Hankerson et al. [2004, chap. 1.3], a 160 bit ECC key offers the same level of security like a 1024 bit RSA key. As for ECC unlike for RSA no sub-exponential time algorithms are known, for higher security levels the advantage of ECC is even higher. Particularly interesting is the security level equivalent to the symmetric cryptography standard AES with 128 bit keys. To achieve this level for ECC suffice 256 bit

keys, while RSA requires 3072 bit. Although RSA currently is more widespread used than ECC, more and more applications take advantage of ECC. For example a large share of the smart cards, called *Bürgerkarte* [buergerkarte] used in the Austrian e-Government system, make use of ECC in 192 bit NIST prime fields.

In this paper a multi-core ECC processor architecture for prime fields is proposed, which fits into a mid-range FPGA (Spartan 3 1500) on a PCI board and thus serves for acceleration of ECC operations on server side systems. Simulation and first synthesis results advise that this system can process nearly about 1800 ECC operations per second on a 192 bit NIST prime field. As far as known to the author it is the fastest ECC processor for FPGAs for prime fields. If a Xilinx Virtex 4 LX200 is used, estimates indicate that more than 10000 ECC operations/second can be performed.

In section 2 a summary of other server-side ECC processors is given.Section 3 on the next page gives an overview about elliptic curve cryptography in general. In Section 5 on page 6 our hardware architecture is presented. Finally Section 7 on page 9 gives preliminary results of simulation and expected hardware performance in comparison with other ECC processors.

## 2 Related work

While there are many server side acceleration processors for RSA, there are much less for ECC. Most of these focus on elliptic curves defined over binary extension fields ($GF(2^m)$), because these are computational easier and so a higher performance is achievable as when prime fields $GF(p)$ are used. See Section 4 on page 5 what difficulties arise when $GF(p)$-fields are used. However, when in an application prime field support is needed, for example in the Austrian e-Government system, these advantage can't be used and the hardware must support $GF(p)$ fields. An ECC processor for $GF(2^m)$ is presented in Wolkerstorfer [2004]. The processor presented in this paper is based on this one, and therefore the basic architecture is similar. However, the support for $GF(p)$ fields complicates the architecture considerable.

One of the fastest ECC-processors for $GF(p)$ is presented in Eberle et al. [2004]. It is a very powerful processor, which can also perform calculations for $GF(2^m)$ and RSA. Using a 64 bit multiplier, it achieves a very high performance, for example 6000 ECC-operations per second for a $GF(p)$-field with 224 bit. However, these scores are for a hypothetical implementation in current processor technology at 1.5 GHz and it is very questionable whether it is feasible to use such powerful and therfore expensive technologies for a cryptographic processor. They also implemented a prototype using a Xilinx Virtex 2 V6000 FPGA running at 66 MHz. Unfortunately no benchmarks are given for this implementation. Assuming that it could run at 100 MHz it would be about 15 times slower than the standard cell implementation. Therefore it can be estimated that even for the smaller $GF(p)$-192 bit fields less than 1000 ECC-operations can be performed per second.

Satoh and Takano [2003] present an ASIC elliptic curve processor which supports both $GF(2^m)$ and $GF(p)$-fields with arbitrary primes and reduction polynomials. The high-speed implementation uses a 64x64-bit multiplier running at a clock rate of 137.7 MHz. It takes 1.44 ms for a point multiplication over a 192 bit prime field. This corresponds to nearly 700 point multiplications per second.

In Crowe et al. [2005] an arithmetic unit is proposed which can handle both RSA and ECC over GF(p). To handle the high difference of typically used field sizes multiple arithmetic units are used, which are pipelined for RSA operations and work in parallel for ECC operations. The architecture uses carry propagate adders, which can be a reason for the relative low clock frequency obtained on the target Xilinx XC2v2000 FPGA of less than 50MHz for 256 bit prime fields. A 256 bit field multiplication takes $5.75\ us$. Estimating the performance for point multiplication cannot be done easily as the multipliers operate in parallel and so performance would depend very much on how well

the point multiplication can be parallelized.

Orlando and Paar [2001] present a processor for $GF(p)$ which has a quite unique architecture using Montgomery multiplication with booth encoding and pre-computation. A prototype was implemented in a Xilinx XCV1000E FPGA and. It uses about 11416 LUTs and due to the pre-computation about 5700 flipflops and 35 block RAMs. It can be clocked at 40 MHz. Only a raw estimate for the performance is given, which is $3ms$ per point multiplications, which corresponds to about 330 point multiplications/second.

Örs et al. [2005] uses montgomery multiplication using a systolic array multiplier. Implemented in a Xilinx Virtex E-1000 6000 slices are occupied, and a maximum clock frequency of over 90 MHz is possible. 160 bit point multiplication time is 14.14 ms, which corresponds to 70 point multiplications per second.

# 3 Mathematical fundamentals

## 3.1 Elliptic curve cryptography

The use of elliptic curves for cryptographic systems was independently proposed by Koblitz [1987] and by Miller [1986]. Later a couple of cryptographic protocols were defined. Among the most important ones is the *Elliptic Curve Digital Signature Algorithm (ECDSA)* which was standardized by various standardization organizations. It is a public key signature scheme, that can be used to sign digital documents with the private key of the signer, while everybody can verify the authenticity of the document by using the public key of the signer. ECDSA plays a key role in the target applications of this work, therefore the hardware was designed with principally this algorithm in mind. However, the base operation of ECDSA, the *point multiplication ($kP$)* is also the base operation in all other elliptic curve cryptography protocols, for example in the encryption scheme *Provably Secure Encryption Curve scheme (PSEC)*. Therefore support for these can be easily added with changes in the software only.

The security of elliptic curve cryptography is based on the difficulty to solve the *Elliptic Curve Discrete Logarithm Problem (ECDLP)*, which is to calculate $l$ in the equation $Q = lP$. $Q$ and $P$ are points on the elliptic curve $E$ over the finite field $F_q$. The operation in $lP$ is called *point multiplication*. No algorithms with sub-exponential run time are known to solve the ECDLP. When sufficiently large fields are used it is infeasible to solve the ECDLP and so elliptic curve cryptography can be assumed to be secure.

The calculation of the point multiplication over the curve $E(F_q)$ involves calculation in two different domains: Elliptic curve arithmetic operations, and the arithmetic operations in the underlying field.

## 3.2 Elliptic curve arithmetic

The basic algorithm to calculate the point multiplication, $kP$, is the *double and add algorithm*. It works analog to the integer exponentiation algorithm. The scalar $k$ is bitwise scanned, and depending on whether the current bit is set, the base point $P$ is only doubled, or doubled and added to the intermediate result. As $k$ is chosen randomly, on average $m$ point doublings and $m/2$ point additions are required to calculate the point multiplication, where $m$ is the number of bits of $k$.

Various optimizations can be performed to reduce the number of elliptic curve operations performed in the point multiplication. Frequently used are *window methods*, which pre-calculate a certain number of point-multiples. This is in particular efficient as the base point for many elliptic curve

algorithms is constant. However, for hardware implementations window methods are less appropriate as the use a lot of memory, which is costly in hardware, in particular in standard cell designs. Additionally, control is complicated.

Another approach, which can also be combined with the window methods, is to convert the scalar $k$ in a special form, which has less bits set, and therefore reduces the number of elliptic curve point additions. Widely used is the *non-adjacent form (NAF)*. Here a bit of *k* can also be negative, which allows reducing the number of set bits to about $m/3$. Using negative numbers is possible, as the point subtraction is as efficient as point addition. The overall performance gain of this measure is not very impressive. However the hardware architecture required for the simultaneous point multiplication, presented in the next chapter, can be reused, and therefore despite of this it makes sense to use NAF for our processor architecture.

The *simultaneous point multiplication* is a special optimization for ECDSA. In the verification step its necessary to calculate the sum of two point multiplications ($kP + rQ$). As the point multiplication is the dominating factor in the ECDSA this leads to a nearly doubling of the calculation time for the verification in comparison to the time required for the signing. To avoid this an algorithm can be used which can do both point multiplications and the addition simultaneously. For this a single pre-calculation of $P + Q$ is required. The main loop of the algorithm is changed, so that it evaluates both scalar numbers $k, r$. When only one current bit of either *k* or *r* is set, the corresponding point is added, when both are set, the pre-calculated point $P + Q$ is added. Trading a slightly more complicated control and memory for the two additional saved points, that is $Q$ and $P + Q$, the runtime is reduced by far. The number of doublings is halved and the number of adds is reduced by one fourth in comparison to calculating both multiplications separately and adding the intermediate results. As already mentioned another advantage is that the hardware for the simultaneous point multiplication can be reused for supporting scalar in NAF when $kP$ has to be calculated in the signing algorithm.

For elliptic curves over binary extension fields $GF(2^m)$ the *Montgomery-method* is a more efficient algorithm to calculate the point multiplication. However, for prime fields $GF(p)$ it is less efficient than the double and add algorithm. Another advantage of the Montgomery method is that in every step both a point addition as a point doubling is performed. This gives a certain protection against *side channel attacks*, which use timing or power analysis to get information about the secret scalar $k$. However, in this work the security of the hardware must only be equal to the security of a pure software system, as in FPGA-technology a secure storage of the private key is not possible. For verification no additional measures are necessary because there is no secret involved. For signing only protection against timing attacks is required because direct access to the hardware which is required for power analysis would allow retrieving the secret key from the FPGA and also the server directly. The protection against timing analysis can be done by adding additional wait cycles in the driver or the ECDSA software.

As already mentioned for the point multiplication *point addition* and *point doubling* is used. Both use a series of operations in the underlying finite field, which are, if the points are given in affine coordinates, multiplications, additions/subtractions and one inversion. As the inversion is a very expensive operation, reducing the number of inversions is very useful. This can be done by using projective coordinates. Here an additional point coordinate is added. At the cost of using about four times more multiplications it reduces the number of inversions to only one, which is performed after the complete point multiplication and converts the result in projective coordinates back to affine coordinates. In this work Jacobean-projective coordinates are used. This allows the point addition of a point in affine coordinates to a point in Jacobean-projective coordinates, thus no conversion of the point $P$ to projective coordinates is required. The overall performance is very dependent on the speed of operations in the underlying field, which are discussed in the next section.

### 3.3 Finite field arithmetic

For the finite field over that the elliptic curve is defined two types of fields are widely used: *Binary extension fields* $GF(2^m)$ and *prime fields* $GF(p)$. For a discussion of advantages of each one see section 4. In the following only prime fields are discussed.

A prime field $GF(p)$ is a finite field with $p$ elements. $p$ must be a prime, otherwise the elements only define a group instead of a field. The field operations addition, subtraction, multiplication and inversion are performed modulo this prime $p$. Therefore a reduction has to be performed after each modular operation, or so that the result always is smaller than an upper bound, which is here defined by the hardware size. Generally, the reduction is a costly operation because a division is needed to estimate the reduced result. To simplify the reduction two approaches are widely used:*Montgomery-multiplication* and *reduction for special primes*.

The first one is the Montgomery-multiplication [Montgomery, 1985]. It trades an additional transformation step, for replacing the trial division with a very easy one, usually a division by a power of two, which can be performed as a simple right shift. As in the point multiplication many consecutive field multiplications are performed, the time for transformation and the back-transformation, both are themselves a Montgomery multiplications, is negligible, and the performance improvement is high.

The second approach waives the support for arbitrary primes and only supports reduction for some specific primes, for example the NIST primes are recommended in FIPS 186-2 standard. This is possible as the cryptographic protocols usually only use these primes. Therefore reduction for arbitrary primes is not required. These primes are selected in a way that allows very efficient reduction. Instead of a trial division only a few additions and subtractions are needed, for example in a 192 bit $GF(p)$ field to perform the reduction only three 192 bit additions are required. In comparison to the Montgomery multiplication, the hardware size is significantly reduced, because only one radix multiplier is required instead of two. As this multiplier is the dominant part of the whole architecture, the hardware size is reduced significantly, which allows the use of either higher radices or using multiple cores, and so highly improves the overall performance.

The field inversion can be implemented with the *extended Euclidean algorithm* or by using the *theorem of Fermat*. The first method is faster but requires extra hardware, while the second approach can reuse the multiplier for calculating $a^{-1} = a^{2^p-2} \pmod p$. This exponentiation takes about $2p$ multiplications, with $m$ the length of the modulus $p$. That is because the primes commonly used do have nearly all bits set. In this work the second approach was chosen.

## 4 Comparison of $GF(2^m)$ and $GF(p)$ fields

This sections gives an overview of the difficulties that arise when $GF(p)$ fields are used instead of $GF(2^m)$ fields.

$GF(2^m)$ fields have the advantageous property that the field addition is just the *xor*-operation. Thus each bit of the result can be calculated independently from other bits of the input number. This means that no carry propagation occurs. In $GF(p)$ fields this is not the case. Here the carry may propagate from the least significant bit up to the most significant bit of the result. As the numbers used in cryptography typically are very large, this carry propagation would lead to a very long critical path in the hardware design, and would result in very low maximal clock frequencies. To cope with this usually *Carry Save Adders (CSA)* are used. These calculate the sum and the carry of three input numbers separately. The three inputs can be for example one number in redundant representation, and another number in binary representation. A tree structure must be used when more than three numbers are added. This, together with the additional register used for saving the carry result, leads

to a substantial larger hardware than when $GF(2^m)$ fields are used. This in particular occurs for high multiplication radices as in the radix multiplier for each additional bit in the radix an additional partial product must be added in the adder tree. When the binary result is needed, this is for example the case when the result of some sequential field operations must be saved in memory, or when the result is needed as an operand for a multiplication, the redundant result must be converted to its binary value by adding them together. This can either be done by a separate adder or by reusing the CSA-adder multiple times. As the first approach requires a large additional hardware, the second approach was used in this architecture. The drawback is that additional cycles are required for calculation. On average $ld(bits)$ cycles are used for converting the result. This time does not depend on the multiplication radix. Therefore the strategy to use higher radices for receiving more performance is inapplicable because the time for the conversion becomes soon the dominant factor. To cope with this two approaches were used. The first one is two use multiple cores instead of using very high radices. The second one is two use a mixture between CSAs and normal adders in the feedback path, where the width of the conventional adders depends on the radix of the multiplier, so that both paths are balanced. This reduces the conversion time for higher multiplication radices, and therefore makes it feasible to use them. See section 5.1 for details.

Another advantage of $GF(2^m)$ fields is that squaring is much simpler than the multiplication of two different numbers, actually it can be done in a single cycle. For a ECC-point-multiplication over an 191 bit $GF(2^m)$-field about 2000 field multiplications and about 1300 field squarings are performed. The time required for a multiplication is about $191/radix$, for example $24$ cycles for radix $8$. Obviously the possibility to calculate the squaring in one cycle saves a lot calculation time, while in $GF(p)$ fields the squaring is not different to a multiplication.

The third and least important advantage is, that the subtraction operation is the same as the addition operation in $GF(2^m)$ fields. This implies that no negative numbers exist. This simplifies the hardware as no sign extension is required. To avoid sign extension, which is in particular unpleasant in the reduction circuit for $GF(p)$ fields a special negation is used which always adds a certain multiple of the prime $p$ to the result. This assures that neither the sum nor the carry part of the result is negative. See section 5 for details.

# 5 Hardware architecture

Figure 2 on the following page shows the architecture of the elliptic curve processor core. As for server applications throughput is the most important benchmark, multiple cores can be used to improve overall performance. The main part of each core is the *ALU* which performs the finite field operations. The main task of the *control unit* is to carry out the point multiplication, and it performs also the elliptic curve operations. The *regfile* is the memory for the operands and intermediate results, but also supports the bus-io-transfers, by supplying, additionally to the full word length, 32-bit-IO-ports with bank select. To reduce area requirements this circuit is reused for retrieving of the current bit of the scalars $k$ and $l$. For this task additionally a 32-1 multiplexor is used.

## 5.1 ALU

The *ALU* (see figure 1 on the next page) operates on the full length word for one operand of the multiplication and for all other operations. The second multiplication operand is processed on a per digit basis with a parameterizable radix size, which can be between 1 bit and 32 bits for 192 bit prime fields. The maximum radix is limited by the reduction unit, which can easily extended to support larger numbers. However, simulation results show that for high radices the redundant-binary-conversion is dominating, and therefore using higher radices is little beneficial.

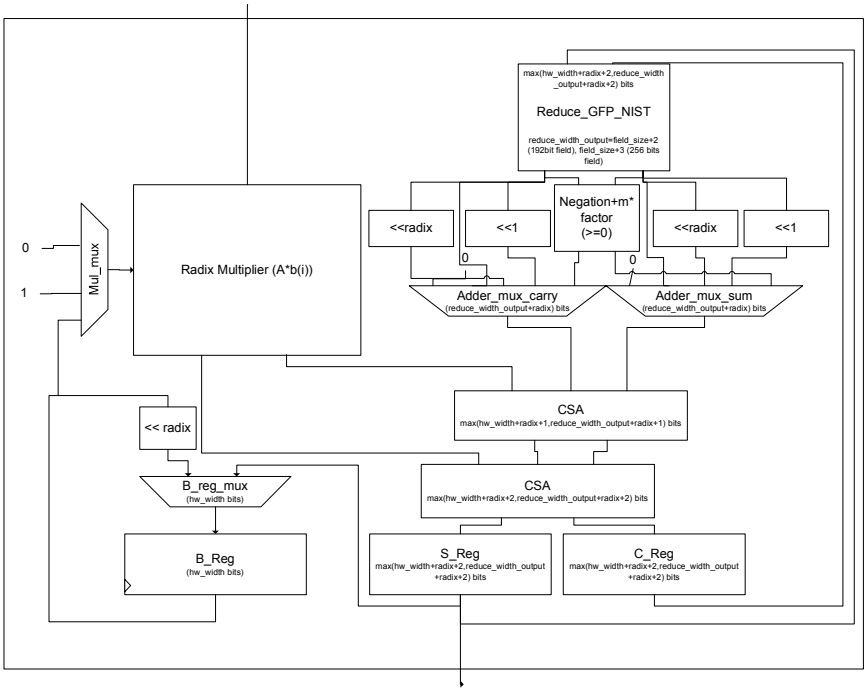


**Figure 1:** Elliptic curve processor ALU

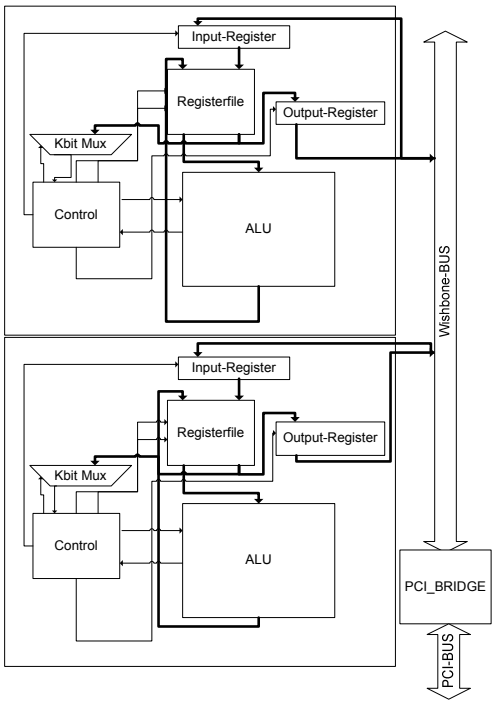


**Figure 2:** Elliptic curve processor architecture

The ALU consists of two paths: The radix multiplier and the feedback loop, which besides the shifting for the multiplication, also performs the other finite field operations, and includes the reduction functionality. The results of both paths - four values because both radix multiplier and feedback loop use numbers in carry save representation - are summed up with two carry save adders and saved in two registers. Locating the reduction unit in the feedback loop does have the drawback that an additional clock cycle is required to calculate the reduced result. However, it has the advantage that critical path is shorter and that the pathes are well balanced. This allows the use of hybrids of CSA and carry propagation adders, where the width of the carry propagation adders is varied with the radix of the radix multiplier, such that both critical pathes stay balanced. The benefit is that the carry-save-redundant-conversion takes less cycles when higher radices are used, which improves the scalability of the architecture for higer radices.

The *radix multiplier* is based on the one presented in Wöckinger [2005]. The results of partial product generators are added up by a *Wallace-tree* using carry save adders. This structure is optimized for a minimum propagation delay. However, it has the drawback that the structure is highly irregular, which leads to complicated routing. Particular for FPGA-Implementations it is probable that using a structure which has a higher delay but is more regular, for example a carry save adder array, performs better.

The *reduction unit* reduces the intermediate result by using the equations for reduction with NIST primes recommended by the FIPS 186-2 standard. A simplified version of the equations was used which only support inputs up to $field - size + k$ instead of $field - size * 2$ with $k = 64$ for the 192 bit-prime field. This saves adder/subtracter steps on the cost of limiting the maximum radix for the multiplier. While the reduction with the 192 bit NIST prime only uses additions, for larger NIST-prime-fields also subtractions are required. To avoid signed numbers the same strategy is used as in the negation unit. A multiple of the NIST prime is added, which is definitely larger than the possible most negative result. This ensures that both the carry and the sum part of the reduction result are always positive. As a drawback the reduction result can now be larger, for example one bit for 256 bit fields, which requires that the hardware width must be enlarged by this single bit, and the final result of the complete point multiplication has to be fully reduced, which is only an integer subtracting of the NIST-prime. To support in an arithmetic unit not only a single field, but also all smaller NIST-prime fields, a multiplexor can be added easily to allow switching the reduction unit for different fields.

The *negation unit* is used for the subtract operation. It calculates the two's complement of the intermediate result, and adds a multiply of the reduction modulos to ensure that both the carry and the sum result after the inversion are always positive. This is required because sign extension, which is required for negative numbers, would complicate the hardware, and is incompatible with the redundant-binary-conversion because of the reduction unit is in the loop-back path.

# 6 Implementation

The target platform for the processor is a Xilinx Spartan 3-1500 PCI Board [AVNET]. The board does not provide a PCI bridge, so the Opencores PCI bridge core is used [Opencores.org]. The multiple elliptic curve processor cores are connected over a wishbone bus with the PCI bridge (see figure 2 on the preceding page). The cores can be parameterized for different field sizes, which is an alternative for supporting different field sizes by the ECC processor itself. The driver for the card is implemented in Linux, and is doing the scheduling for the multiple cores. This approach keeps the hardware small, and should only cause negligible performance losses, because despite the high performance of the elliptic curve cores, the absolute number of point multiplications performed per second is still very low. The Java Elliptic Curve Cryptography Library [IAIK] was extended to use the hardware for

| Reference | $GF(p)$-field | Multiplier architecture | Target | Area | Freq. (MHz) | Cycles per $kP$ | $kP/s$ |
|---|---|---|---|---|---|---|---|
| This work | NIST-192 | two 192x8 | Spartan3-1500 | 12000 Slices | 100 | 112400[1] | 1779 |
| This work | NIST-192 | five 192x32[2] | Virtex4-LX200 | 70000 Slices | 100 | 49150[1] | 10100 |
| Satoh and Takano | 192 | 64x64 | $0.13um$ ASIC | 118000 Gates | 137.7 | 198288 | 694 |
| Eberle et al. | 224 | 64x64 | ASIC[3] | ? | 1500 | 245330 | 6114 |
| Eberle et al. | 224 | 64x64 | Virtex2-6000 | ? | 66 | 245330 | 270[4] |
| Örs et al. | 160 | systolic array | VirtexE-1000 | 6055 Slices | 91.3 | 1316160 | 69.3 |

**Table 1:** Comparison ECC-$GF(p)$-processors

computing elliptic curve operations over finite fields supported. This allows that application using the library can transparently use the hardware accelerator. As test application amongst others an implementation of the ECDSA algorithm is used.

# 7 Results and Conclusion

The project is currently in an advanced stage. More precisely the ALU was fully implemented in VHDL and tested in simulation, the control is largely finished. The performance estimates are based on the results of a cycle-accurate high level language model of the processor, whose results were verified with VHDL-simulation results. Synthesis runs were performed to retrieve estimations of the timing and area requirements of the processor. Therefore the estimates are expected to be sufficiently accurate. The main factor of uncertainty is how many instances of the processor finally will fit into the FPGA. Table 1 shows that the processor performs very well in comparison with other implementations. It is s much faster than the reported FPGA results and competes very well with the ASIC implementations. This is achieved by the restriction of only supporting the essential fields. Most other implementations focus on supporting arbitrary fields, which can be useful when they are used for small client devices. However, for server applications, which are targeted in this work, the benefit is arguable because usually virtually all ECC-operations to perform will be over NIST-prime-fields. Thus the few others can be performed by the main processor. The performance loss for this can be safely expected to be much less than the performance loss of the hardware caused by the requirement to support arbitrary prime fields. However, dual field support, that is supporting $GF(p)$ and $GF(2^m)$ fields, could be - depending on the application - a useful extension of the ECC-processor, because this requires only little additional area.

[1] Effect of carry-save-carry-propagate hybrid adders (see 5.1) is not considered. In particular for the 32 radix size a better actual performance can be expected

[2] Max. core number limited to five by PCI-bridge implementation. Actually using smaller radices and more cores would be more efficient. (for example 14000 $kP/s$ for ten 16-bit radix cores)

[3] Current processor technology which allows 1.5GHz when architecture is fully pipelined

[4] Author's estimate based on reference

## References


AVNET. Spartan 3 1500 evaluation kit. URL http://www.em.avnet.com/evk/home/0,1719,RID%253D0%2526CID%253D7816%2526CCD%253DUSA%2526SID%253D4746%2526DID%253DDF2%2526SRT%253D1%2526LID%253D0%2526PVW%253D%2526BID%253DDF2%2526CTP%253DEVK,00.html.

buergerkarte. Bürgerkarte. URL http://www.buergerkarte.at/.

Francis Crowe, Alan Daly, and William P. Marnane. A scalable dual mode arithmetic unit for public key cryptosystems. In *ITCC (1)*, pages 568–573. IEEE Computer Society, 2005.

Hans Eberle, Nils Gura, Sheueling Chang Shantz, Vipul Gupta, Leonard Rarick, and Shreyas Sundaram. A public-key cryptographic processor for rsa and ecc. In *ASAP '04: Proceedings of the Application-Specific Systems, Architectures and Processors, 15th IEEE International Conference on (ASAP'04)*, pages 98–110, Washington, DC, USA, 2004. IEEE Computer Society. ISBN 0-7695-2226-2.

Darrel Hankerson, Alfred Menezes, and Scott Vanstone. *Guide To Elliptic Curve Cryptography*. Springer Professional Computing. Springer, 2004.

IAIK. Iaik elliptic curve crytography library. URL http://jce.iaik.tugraz.at/products/17_ecc/index.php.

Neal Koblitz. Elliptic curve cryptosystems. In *Mathematics of Computation, 48*, pages 203–209, 1987.

Victor S Miller. Use of elliptic curves in cryptography. In *Lecture notes in computer sciences; 218 on Advances in cryptology—CRYPTO 85*, pages 417–426, New York, NY, USA, 1986. Springer-Verlag New York, Inc. ISBN 0-387-16463-4.

Peter L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, April 1985. ISSN 0025-5718.

Opencores.org. Opencores pci bridge. URL http://www.opencores.org/projects.cgi/web/pci/home.

Gerardo Orlando and Christof Paar. A scalable gf(p) elliptic curve processor architecture for programmable hardware. In *CHES '01: Proceedings of the Third International Workshop on Cryptographic Hardware and Embedded Systems*, pages 348–363, London, UK, 2001. Springer-Verlag. ISBN 3-540-42521-7.

Akashi Satoh and Kohji Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Trans. Comput.*, 52(4):449–460, 2003. ISSN 0018-9340.

Johannes Wolkerstorfer. *Hardware Aspects of Elliptic Curve Cryptography*. PhD dissertation, Graz University of Technology, Austria, Institute for Applied Information Processing and Communications, July 2004.

Thomas Wöckinger. High-speed rsa implementation for fpga platforms. Master's project, Graz University of Technology, Austria, Institute for Applied Information Processing and Communications, January 2005.

S. B. Örs, L. Batina, B. Prenel, and J. Vandewalle. Hardware implementation of an elliptic curve processor over gf(p) with montgomery modular multiplier. *International Journal of Embedded Systems (IJES)*, page 15, 2005.