Rapid Prototyping for Hardware Accelerated Elliptic Curve Public-Key Cryptosystems

M. Ernst, S. Klupsch, O. Hauck, and S. A. Huss
{ernst | klupsch | hauck | huss }@iss.tu-darmstadt.de
Integrated Circuits and Systems Lab, Computer Science Department
Darmstadt University of Technology, Germany

Abstract

A generator-based design and validation methodology for rapid prototyping of elliptic curve public-key cryptosystem hardware is described. By their very nature, crypto systems challenge both design and validation. Pure RTL-based synthesis is as unsuitable as is high-level synthesis. Instead, a generator program accepts the two main parameters, key size and multiplier radix, and creates a highly efficient custom RTL description which is synthesized into a FPGA.

This approach benefits the design in that it allows to effortlessly exploit the available resources on the FPGA for variable requirements of security and performance. It is also advantageous for validation of the correctness of the design as for small parameter values the design can be tested exhaustively. Thus, the correctness for large key sizes depends only on the correctness of the generator.

Furthermore, deploying FPGAs supports integration of an ASIC realisation of the same algorithm which boosts performance. By emulating its interface, the ASIC can be accomodated even before fabrication thus enabling mixed FPGA/ASIC acceleration of elliptic curve cryptosystems.

1 Introduction

Many E-Commerce applications are characterized, for instance, by their demand for confidential data exchange via public communication networks (e.g., Internet). These data exchanges must be protected from fraudulent access by third parties. The basic technology which can warrant this kind of protection is known as Public-Key Cryptography.

Besides the widely-used RSA method, public-key methods based on elliptic curves (EC) have gained more importance because they are believed to give higher security per key bit, i. e. one can work with shorter keys [1] (1024 RSA-bits are equivalent to 160 EC-bits). The smaller key size permits a more cost-efficient implementation and higher throughput.

The operation of point multiplication ($k{\cdot}P$) is the basic arithmetic in the area of EC cryptography. This is a complex operation, and its computation is very time consuming. Basically, the time required for the computation of $k{\cdot}P$ determines the performance of EC implementations. The *Elliptic-Curve CryptoProcessor* presented in the following implements this operation within hardware. The performance of EC cryptosystems can be enhanced significantly by the use of this processor.

The mathematical background for the CryptoProcessor is explained in the following section. Sections 3 and 4 deal with specification and validation for this crypto application, respectively. Section 5 briefly explains the ASIC implementation, while section 6 details the FPGA implementation and our prototyping environment, followed by conclusions.

2 Public Key Cryptosystems

Common to all public-key schemes is a computationally hard problem on which the security of the cryptosystem is based. In RSA it is the factoring problem, meaning that it is easy to multiply two large primes but impossible with current techniques to factor a 1024-bit number which is a product of two large primes. The other problem whose intractability is exploited in this manner is the so-called discrete logarithm problem in large finite groups. This refers to the fact, that, given a large finite group G, group elements $P, Q \in G$ and $k \in N$ with $Q = k{\cdot}P$, it is hardly possible to compute k given only P and Q. The reason for the name discrete *logarithm* stems from the fact that one notates the abstract group operation either additively, as before, or multiplicatively as $P^k = Q$. In the latter notation k is the logarithm of Q to the base P.

In the following sections the basics of elliptic curves are introduced and the algorithm for the computation of $k{\cdot}P$ is described.

2.1 Mathematics of Elliptic Curves

In the sequel we will consider so-called non-supersingular elliptic curves only as they provide the highest security. Non-supersingular elliptic curves are defined as the set of solutions of the cubic equation

$$y^2 + xy = x^3 + ax^2 + b \qquad (1)$$

1074-6005/01 $10.00 © 2001 IEEE

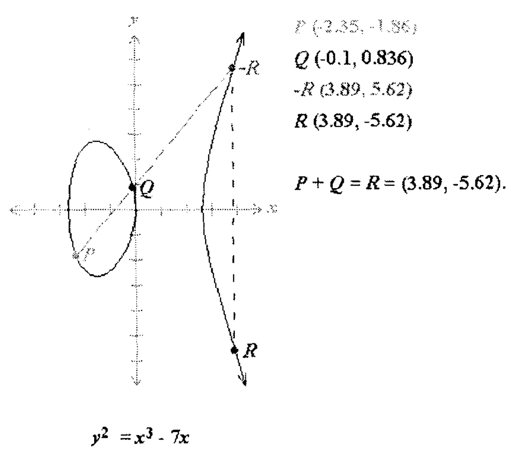

Figure 1: Example of an elliptic curve over the real numbers visualizing the point addition.

together with the point at infinity $\mathcal{O}$, where $a, b \in F, b \neq 0$, and F is a field of characteristic two. By defining an appropriate addition operation an elliptic curve becomes an abelian group. Thus, by varying the parameters a, b one has a source of finite abelian groups. Fig. 1 shows an example of an elliptic curve over the reals. Here, a geometric interpretation of the addition can be given: Find the third intersection point (-R) of a straight line through P and Q with the elliptic curve. The result $R = P + Q$ is found by mirroring -R at the x-axis.

For non-supersingular elliptic curves the basic operation of adding points $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ is as follows:

$$R = P + Q = (x_3, y_3)\,. \tag{2}$$

If $P \neq Q$ (addition), then

$$\phi = \frac{y_2 - y_1}{x_2 - x_1} \tag{3}$$

$$x_3 = \phi^2 + \phi + x_1 + x_2 + a \tag{4}$$

$$y_3 = \phi(x_1 + x_3) - y_1 + x_3\,. \tag{5}$$

If $P = Q = (x, y)$ (doubling), then

$$\phi = x + \frac{y}{x} \tag{6}$$

$$x_3 = x^2 + \frac{b}{x^2} \tag{7}$$

$$y_3 = x^2 + (\phi + 1)x_3\,. \tag{8}$$

To avoid computing inverses, we switch to projective coordinates, where $x^* = \frac{x}{z}$ and $y^* = \frac{y}{z}$ are the corresponding affine coordinates. If $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$, where $P, Q \neq \mathcal{O}$ and $P \neq -Q$, then $R = P + Q = (x_3, y_3, z_3)$ is given for $P \neq Q$ by:

$$x_3 = AD \tag{9}$$

$$y_3 = CD + A^2(Bx_1 + Ay_1) \tag{10}$$

$$z_3 = A^3 z_1 z_2\,, \tag{11}$$

where $A = x_2 z_1 + x_1 z_2, B = y_2 z_1 + y_1 z_2, C = A + B$, and $D = A^2(A + a z_1 z_2) + z_1 z_2 BC$. We can fix z_2 to 1, as $P_0 = (x_0, y_0, 1)$ will always be added.

If $P = Q$, then

$$x_3 = AB \tag{12}$$

$$y_3 = x_1^4 A + B(x_1^2 + y_1 z_1 + A) \tag{13}$$

$$z_3 = A^3\,, \tag{14}$$

where $A = x_1 z_1, B = b z_1^4 + x_1^4$.

Thus, computing $P + Q$ (*EC-Add*) requires 13 finite field multiplications, 7 additions and 1 square operation. The computation of $2 \cdot P$ (*EC-Double*) requires 7 multiplications, 4 additions and 5 squares. This is also illustrated in the lower part of Fig. 2.

Ultimately, multiplication, addition and squaring have to be done in the underlying finite field (FF). In our case we use F_{2^n} as the underlying FF. Exploiting a field of characteristic two reduces the addition to XOR-ing the corresponding bits. This can be done very efficiently in hardware. The elements of F_{2^n} are represented in the so-called Optimal Normal Basis (ONB) so that squaring can be easily performed by a cyclic shift. For multiplication the Massey-Omura architecture [2] is used. In this architecture all bits of the result vector can be computed independently from each other, so that very area-efficient bit-serial implementations as well as high-performance parallel implementations are feasible. A more detailed description of ONBs and how multiplication is done in ONB representation can be found in [3] and [4].

2.2 Algorithm

The operation $k \cdot P$ is performed with the *Double-and-Add* algorithm (see Fig. 2) as repeated doublings and addings of the base point P_0 on the elliptic curve using the underlying EC arithmetic, which was described in the previous section. Each $k \cdot P$ multiplication requires n *EC-Double* and h *EC-Add* operations. As *EC-Double* is cheaper in terms of multiplication as *EC-Add*, the performance of the algorithm benefits from a key k with low Hamming weight h. The EC arithmetic in turn is based on the underlying finite field arithmetic. Here, as already mentioned, the FF operations *FF-Add* and *FF-Square* can be implemented very efficiently, so that the performance of the EC arithmetic is mainly determined by the time required for the computation of *FF-Mult*. For one $k \cdot P$ computation $7n + 13h$ *FF-Mult* operations have to be done. Using the previously described multiplier architecture this results in $n(7n + 13h)$ Massey-Omura elementary operations taking about $\log n$ time each.

Example: For $n = 270$ and $h = 40$ with a pure bit-serial implementation of the Massey-Omura multiplier, one computation of $Q = k \cdot P$ requires 650700 multiplier iterations.

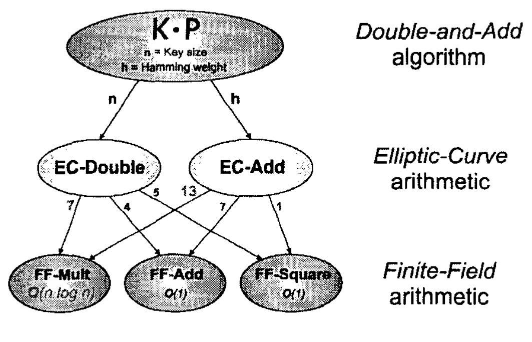

Figure 2: *Double-and-Add* algorithm

We picked $n = 270$ as key size for our hardware implementation because of the existence of an ONB for it. From theory this is more secure than RSA with 2048 bits. Previous implementations [5] use just 155 bits for the key.

3 Design Specification

In this paper we focus on an EC crypto algorithm, which is realized in a single integrated circuit. This application, as many other digital circuits, is based on models in a hardware description language (VHDL).

3.1 Modelling Levels

VHDL descriptions can be processed by synthesis tools to derive a netlist of basic logic elements, which can then be fed into place and route tools. Based on this design flow *Register Transfer Level* (RTL) descriptions have proven to be well suited to efficiently design integrated circuits. In addition to using commercial synthesis tools, there is a lot of potential for application specific model generators, which build an RTL description from a more abstract rule set.

RTL is a good choice when aiming at synthesizable models which have to be independent from the utilized synthesis tool. In 1999 an IEEE Standard [6] was drafted which defines a rule set for RTL synthesis. RTL is characterized by functional blocks, such as registers, memory units or ALUs, and control logic. The control logic is based on clocked state transitions. This abstraction level is well-suited for synchronous designs with a clear functionality and hard chip-size or performance constraints. Designers have a good chance to add manual optimizations, and complexity can be managed by a hierarchical design.

The *Algorithmic Level* is defined with multi-cycle operations in mind. While each control step in RTL is based on a clock cycle, the algorithmic level uses a causality paradigm. This results in descriptions which are similar to software programming languages. The number of clock cycles needed for evaluating an algorithm in hardware can be left open. The synthesis tool has to provide for resource allocation, scheduling, and pipelining of datapath and control logic. Code reuse is encouraged and design validation is simplified, but thoroughly checked RTL descriptions can be more efficient. For applications such as the proposed CryptoProcessor the most important optimization goals are throughput and area. The modeling of multi-cycle operations is not of primary concern. To overcome the deficiencies of current algorithmic synthesis tools, it has become clear that a different approach is needed.

3.2 VHDL Model Generator

The *Generator Approach* is based on the idea of defining an abstract specification which can be processed by a generator program in order to produce a synthesizable hardware description. In our case this specification can be parameterized by the order of the underlying field, i.e. the key size n, and the number of single bit Massey-Omura multipliers, i.e. the radix of the design. The generator includes a *Meta-Model* of the CryptoProcessor, which is independent from these parameters. This meta-model in turn is composed of sub-models, which are corresponding to the functional blocks in the CryptoProcessor's architecture.

The architecture of the CryptoProcessor is introduced here, in order to explain how the generator works and to give the motivation, why a generator is needed to automate the transformation from the abstract specification to a specific bit-level implementation.

The top-level architecture is shown in Fig. 3. It consists of 3 main functional blocks. The *Register File* comes with 16 n-bit registers to hold finite field elements or n-bit integers. The *Controller* is realized as a finite state machine and implements the *Double-and-Add* algorithm. The *FF arithmetic* implements the operations *FF-Mult*, *FF-Add* and *FF-Square*. The operation *FF-Mult* again is composed of at least one Massey-Omura single-bit multiplier. Such a multiplier takes two n-bit inputs and reduces them in a huge XOR-tree to one single bit. Because the structure of this XOR-tree is changing with the underlying finite field, it is impossible to create a generic synthesizable VHDL model for this component. To overcome this deficiency, it has become clear that a custom VHDL model generator is the right approach to this problem.

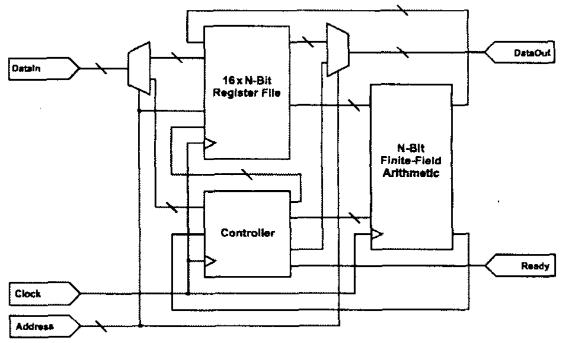

Figure 3: Generic architecture of the CryptoProcessor

Exploiting such a generator we are able to construct CryptoProcessors for various key sizes. Especially when targeting reconfigurable logic devices (FPGA) there are some additional benefits from the generator approach. The available FPGA resources can be used at an optimum, because the number of multipliers (i.e., radix) is not fixed. This in turn is the prerequisite to achieve maximum performance from a specific FPGA. Since the generated VHDL descriptions are not bound to a distinct FPGA family, the rapid advance in FPGA technology can be directly transferred into a better performance. Furthermore, the use of reconfigurable logic enables EC methods with variable key sizes on the same hardware.

The synthesis flow for such a implementation is a two-stage process. First, the model generator produces a VHDL description of the CryptoProcessor for a given key size. Second, this model is fed into an automated synthesis tool to generate the FPGA core.

With respect to design quality and validation there is another benefit from the proposed approach. An implementation for a small key size (e.g., 18 bit) can be used for exhaustive tests. This is mandatory to ensure that the model generator code is correct. For real world crypto applications key sizes of $n \geq 160$ are required, but at this order of magnitude really exhaustive tests are not feasible.

4 Design Validation

A major problem in complex designs is the definition and verification of proper system requirements. In general, it is not possible to obtain a comprehensive specification before the implementation phase has started. Therefore, system requirements must be updated during the implementation phase. Inconsistencies must be mended and functional, temporal as well as structural details must be established. Writing extensive testbenches is a tedious work, which is sometimes evaded by designers who neglect the risks. In the authors' opinion, writing testbenches does not need to cost extra design time. It is even possible to reduce the design time by using testbenches properly. But, in general, it is too late to start working on testbenches if the design is finished.

As a general rule, it is best to keep the design as simple as possible, but there are exceptions. Modeling designs in a generic way enhances the reuseability of components, and it can be used to accelerate testing. However, many applications, including the presented CryptoProcessor, require components that operate on large bit vectors. This makes validation of synthesis results difficult and time-consuming due to the large amount of simulation elements.

The complexity often can be reduced by scaling the signal vectors down. Adding such flexibility is an excess work, but it pays off. On one hand it allows the reuse of the component in contexts with different accuracy requirements, on the other hand it unveils large speedups in functional testing.

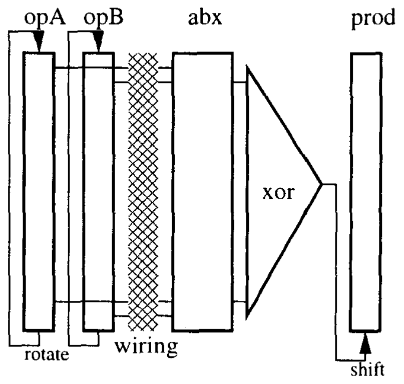

Figure 4: Architecture of the AWP multiplier

The generator approach allows to keep the design entry simple, whilst adding support for a choosable key size. Exception handling and algorithmic tuning can be based on such scaled components, even though final tests might rely on the original component.

This concept was exploited for designing and validating the CryptoProcessor algorithms. In this application, the validation task is twofold. On the one hand, the functionality of the silicon has to be tested with respect to the RTL models. On the other hand, we have to validate the RTL models themselves. In other words, it has to be shown that the EC and the FF arithmetics are reproduced correctly in the hardware's behavioral models. Therefore, the confidence in the correctness of the generator is all-important.

The generator is used to produce implementations for finite fields with small key size which can be verified by exhaustive testing. By validating independent implementations for finite fields with a small key size, it is assumed that the specifications in the generator are faultless and that a coarse validation for the 270-bit key implementation[1] will suffice. Therefore it has to be ensured that the generator specification is well suited to derive RTL models for all interesting key sizes. This has been a major objective during the development of the CryptoProcessor.

5 ASIC Implementation

Due to the immense computational effort, high-performance hardware implementations are necessary in order to enable the use of EC methods in server-based cryptosystems (e.g., online banking servers). The performance of software implementations is not sufficient for this kind of applications because the n-bit FF operations have to be mapped to a processor with fixed word length (e.g., Intel Pentium, 32 bit). For detailed information about leading software implementations we refer to [7] and [8].

[1]The prototyping environment allows to evaluate 10^7 - 10^8 test vectors per day.

A full-custom ASIC version of the CryptoProcessor is currently under design at the author's site [3]. It has two features worth mentioning. First, it is fully asynchronous. The control consists of three state machines on finite field, elliptic curve, and *Double-and-Add* level, respectively. By the nature of the algorithm, those FSMs differ in transition rate by orders of magnitude, i.e., the finite field level is running at GHz speed while *Double-and-Add* rate is below 1 MHz. Self-timed control adapts to the different speeds and thus saves power and improves performance.

Second, the proprietary asynchronous wave pipeline (AWP) circuit technique [9] speeds up the multiplier. Its architecture is depicted in Fig. 4. The contents of the operand registers are rotated in each step and are then fed into wiring implementing the ONB. The shuffled operand bits are combined in parallel into the block labelled *abx*, which performs AND and XOR operations. Finally, the n outputs of *abx* are compressed in a XOR tree into one single bit which is shifted into the result register labelled *prod*. Conventional synchronous pipelining would result in a significant latch/clock/precharge overhead due to the large width of the datapath. AWP is a dynamic circuit style which needs neither precharge signals nor explicit latches. Data is propagating as pulsed waves through the logic. As soon as one wave is fully absorbed by the logic, the next data wave can be applied without waiting for clock or precharge signals.

Data rates exceeding 1 GHz can be achieved in a 0.35 μm CMOS technology. A finite field multiplication thus takes only about 270 ns for a key size $n = 270$. A first prototype using a bit-serial multiplier is nearing completion and is expected to deliver about 2300 $k{\cdot}P$ operations per second on approx. 4 mm^2 silicon area. Even if we are using the simplest version of the *Double-and-Add* algorithm, this counts for a speedup of about 12 compared to the highly optimized, leading software implementation presented in [7][2].

6 FPGA Implementation and Prototyping Environment

As mentioned in the previous section, we are implementing a full-custom ASIC which will act as a 270-bit EC CryptoProcessor. Before the ASIC is taped out, one needs to do a lot of validation and system integration work. This scenario is typical to rapid prototyping applications, but common design flows seem to be not suitable. For system integration purposes the ASIC has a simple synchronous interface, so that an In-Circuit-Emulator (ICE) can be used to substitute the ASIC for system test purposes.

The validation task is more complex. Its objective is to ensure that both the hardware descriptions are mapped correctly and that the timing is matching. Furthermore, we have to validate the hardware models themselves.

[2]The times of [7] can't be compared directly, because of different key sizes.

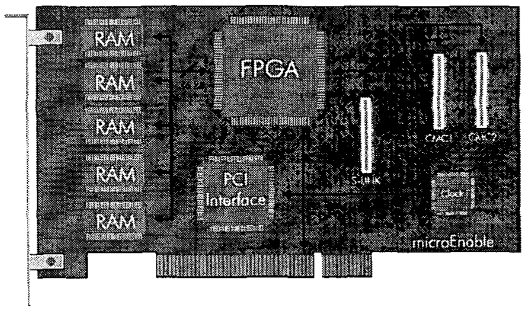

Figure 5: *microEnable* PCI card

A second version of the CryptoProcessor was developed mainly for this purpose. It is based on the *microEnable* PCI card (see Fig. 5) from Silicon Software GmbH [10]. This card is equipped with a FPGA from Xilinx, Inc. [11], a programmable clock generator, static RAM and external interfaces. It is well-suited to be used as an prototyping environment for the full-custom ASIC. Instead of the ASIC another *microEnable* card can be used as an ICE. The integration into a target system is accomplished easily via the PCI interface.

Furthermore, *microEnable* can be used as a stand-alone CryptoProcessor. The use of a FPGA based implementation as the final product can be reasonable if there are no special requirements according to power dissipation or chip size and if the production number of units is small, so that an ASIC implementation would be too expensive. *microEnable* is available with FPGAs of different complexities. A XC4085XLA-FPGA with a complexity of max. 180000 system gates was used to implement the FPGA based stand-alone CryptoProcessor.

The proposed generator approach on top of a VHDL based design flow leads to a 270-bit CryptoProcessor design including 3 Massey-Omura single-bit multipliers. This design has a CLB utilization of 82% and is running at 34 MHz. The resulting performance is 146 $k{\cdot}P$ operations per second.

Two other versions of the CryptoProcessor supporting a key size of 191- resp. 155-bit have been mapped via generator to the same FPGA. This shows how the generator allows to easily trade security for performance. The 191-bit version of the Processor with radix 5 has a CLB utilization of 69% and is running at 36 MHz. The resulting performance is 431 $k{\cdot}P$ operations per second. In contrast, the 155-bit version with radix 7 is running at 37 MHz and has a CLB utilization of 63%. This results in 775 $k{\cdot}P$ operations per second. Note that the achievable CLB utilization decreases when the radix increases. This is because of the relatively poor routing resources a XC4085XLA provides compared to its logic complexity. This problem can be avoided when aiming at recent FPGA architectures (e.g., Xilinx Virtex Family). It is also possible to implement a 191-bit Processor with radix 9 in a XC4085XLA. This results in a CLB utilization of 95%,

Target Platform	Key Size	$k{\cdot}P$ Operations per second
FPGA Hardware [12] (XCV300, 36 MHz)	155	148
FPGA Hardware [12] (XCV300, 33 MHz)	281	70
FPGA CryptoProcessor (XC4085XLA, 37 MHz)	155	775
FPGA CryptoProcessor (XC4085XLA, 36 MHz)	191	431
FPGA CryptoProcessor (XC4085XLA, 34 MHz)	270	146
ASIC CryptoProcessor (AWP, 1 GHz)	270	2300 (estimated)

Table 1: Figures of merit

but the achievable operating frequency decreases to 12 MHz. Therefore, the overall $k{\cdot}P$ performance is lower as in case of an implementation with radix 5. This tradeoff, which has to be done whenever the target FPGA changes, is uniquely supported by the proposed generator approach.

The implementation of a cryptographic processor described in [12] is based on a Xilinx Virtex XCV300-FPGA with a complexity of about 320000 system gates and is seemingly quite similar to our approach: VHDL code is being generated automatically. A more detailed analysis unveils a parameterization of the key size only. Parallelization is not supported, which is essential in order to achieve maximum performance from a specific FPGA. This is the main reason for efficiency stemming from the proposed design method. Even when using a considerably smaller and slower FPGA device for a 155-bit version of the CryptoProcessor the detailed generator approach still achieves a 5 times better $k{\cdot}P$ performance (see Tab. 1).

The figures of merit summarized in Tab. 1 were measured in real-time within our test environment consisting of a standard PC (Intel Pentium III, 550 MHz) running MS Windows NT 4.0. The test application is written in C++ and compiled with MS Visual C++ 6.0. For the hardware synthesis we exercised the FPGA Compiler II V3.5 from Synopsys, Inc. The FPGA mapping was performed using the Foundation Series Software V2.1i from Xilinx, Inc.

7 Conclusion

The acceleration of elliptic curve cryptosystems by ASIC and FPGA hardware poses several unique challenges which are best met by a generator-based design and validation methodology. First, conventional RTL synthesis is not feasible as ONB hardware cannot be cast into a generic VHDL model. Second, due to the large width of the operands, algorithmic synthesis is likely to yield suboptimal results. More importantly, we note that two main parameters determine the design, namely key size and multiplier radix. Therefore, a generator that produces a RTL description for specific parameter values incorporates the best of algorithmic and RTL synthesis for this special application. We thus have the freedom to choose the main parameters to best exploit the FPGA resources while being assured that the optimal hardware structure is being synthesized. If a large FPGA can be afforded, it was shown that competitive solutions can be achieved from FPGAs only.

As an additional asset, FPGAs help in incorporating ASICs as they can emulate their interface beforehand and thus speed up system integration when the ASIC becomes available.

References

[1] N. Koblitz, "Elliptic Curve Cryptosystems", *Mathematics of Computation*, 48, pp. 203-209, 1987.

[2] J. Massey and J. Omura,"Computational Method and Apparatus for Finite Field Arithmetic", *U.S. Patent 4,587,627*, 1986.

[3] O. Hauck, A. Katoch and S. A. Huss, "VLSI System Design Using Asynchronous Wave Pipelines: A 0.35μm CMOS 1.5 GHz Elliptic Curve Public Key Cryptosystem Chip", *Proc. IEEE ASYNC 2000*, Eilat, April 2000.

[4] M. Rosing, "Implementing Elliptic Curve Cryptography", Manning Publications Co., Greenwich, 1999. ISBN 1-884777-69-4

[5] S. Sutikno et al., "Design and Implementation of Arithmetic Processor $F_{2^{155}}$ for Elliptic Curve Cryptosystems", *Proc. Asia Pacific Conference on Circuits and Systems*, pp. 647-650, 1998.

[6] IEEE Standard 1076.6; *Standard for VHDL Register Transfer Level Synthesis*, IEEE Standards Department, New York, 1999.

[7] D. Hankerson, J. Hernandez and A. Menezes, "Software Implementation of Elliptic Curve Cryptography over Binary Fields", *Proc. CHES 2000*, LNCS 1965, pp. 1-24, 2000.

[8] E. De Win, S. Mister, B. Preneel and M. Wiener, "On the Performance of Signature Schemes based on Elliptic Curves", *Proc. Algorithmic Number Theory Symposium III*, LNCS 1423, J. P. Buhler, Ed., Springer-Verlag, pp. 252-266, 1998.

[9] O. Hauck, M. Garg, and S. A. Huss, "Efficient and Safe Asynchronous Wave-Pipeline Architectures for Datapath and Control Unit Applications", *Proceedings 9th Great Lakes Symposium on VLSI*, pp. 38-41, Ann Arbor, March 1999.

[10] Silicon Software, "microEnable Users Guide", 1999.

[11] Xilinx, "Programmable Logic Data Book", 1999.

[12] K.H. Leung, K.W. Ma, W.K. Wong, and P.H.W. Leong, "FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor", *Proc. IEEE FCCM 2000*, pp. 68-76, Napa Valley, 2000.