FPGA Implementation of a Microcoded Elliptic Curve Cryptographic Processor

K.H. Leung, K.W. Ma, W.K. Wong and P.H.W. Leong
{khleung,kwma,wkwong1,phwl}@cse.cuhk.edu.hk
Department of Computer Science and Engineering
The Chinese University of Hong Kong
Shatin, N.T. Hong Kong

Abstract

Elliptic curve cryptography (ECC) has been the focus of much recent attention since it offers the highest security per bit of any known public key cryptosystem. This benefit of smaller key sizes makes ECC particularly attractive for embedded applications since its implementation requires less memory and processing power. In this paper a microcoded Xilinx Virtex based elliptic curve processor is described. In contrast to previous implementations, it implements curve operations as well as optimal normal basis field operations in F_{2^n}; the design is parameterized for arbitrary n; and it is microcoded to allow for rapid development of the control part of the processor. The design was successfully tested on a Xilinx Virtex XCV300-4 and, for $n = 113$ bits, utilized 1290 slices at a maximum frequency of 45 MHz and achieved a thirty-fold speedup over an optimized software implementation.

1 Introduction

Elliptic curve cryptography (ECC) was proposed by Koblitz [1] and Miller [2] in 1985. Compared with other commonly used public key cryptosystems such as RSA and discrete logarithm, claims have been made that ECC offers a smaller key length, better security, and a smaller hardware realization than other methods.

ECC is particularly suitable for embedded applications, the benefits being

- ECC offers the highest security per bit of any known public key cryptosystem so a smaller memory can be used
- ECC hardware implementations use less transistors, as an example, a VLSI implementation of 155-bit ECC has been reported which uses only 11,000 transistors [3] compared with an equivalent strength 512-bit RSA processor which used 50,000 transistors [4]
- ECC is probably more secure than RSA, the largest RSA and ECC challenges solved being 512-bit and 97-bit respectively. In cracking the 97-bit ECC problem, approximately twice the computing power of the RSA problem was used.

Previous implementations of ECC processors have been based on VLSI chips which implement a coprocessor for performing the underlying field operations. An optimal normal basis multiplier over $F_{2^{593}}$ was reported in 1988. It used 90,000 transistors and occupied 0.3 inches on a side. This chip together with a Motorola DSP 56000 was used to implement a complete ECC system which could calculate 5 points a second on a supersingular curve [3]. In 1993, the same team developed a processor for operations in the Galois field $F_{2^{155}}$ [3] which used 11,000 transistors and could operate at 40 MHz. This implementation was intended to be compact yet secure.

A field programmable gate array (FPGA) based processor for elliptic curve cryptography in a composite Galois field $F_{(2^n)^m}$ was developed by Rosner [5]. A compact super-serial multiplier for FPGAs which trades off performance for area was reported in 1999 and its performance for field (polynomial basis) and curve multiplications over $F_{2^{167}}$ has also been presented [6].

In this paper, a Xilinx Virtex based FPGA implementation of an elliptic curve processor is described. Previous implementations based on Galois field processors have the disadvantage that a high bandwidth interface is required to supply the coprocessor with its data. Another limitation of previous ASIC designs is that the field operations are restricted to certain

0-7695-0871-5/00 $10.00

groups (i.e. the subfield and extension fields of $F_{2^{155}}$) and these cannot be changed without fabricating a new chip.

The implementation described in this paper differs from previous implementations from that it has the following features

- the higher level curve operations as well as the field operations are implemented on the chip. This makes the I/O bandwidth requirements much lower than for chips which only implement the field operations
- the curve operations are implemented as sequences of field operations which are programmed in microcode. This allows many algorithmic optimizations to the design to be made without changing the hardware
- the entire design is generated by a module generator which can generate arbitrary key size ECC systems. Thus ECC systems of arbitrary size over an optimal normal basis can be generated (provided they fit on the FPGA device).

The rest of this paper is organized as follows. Section 2 is an introduction to some of the mathematical concepts needed to understand elliptic curve cryptography. The architecture and implementation details of the elliptic curve processor is presented in Section 3. Results are presented in Section 4 and conclusions are drawn in Section 5.

2 Background Mathematics

In an attempt to make this section more readable to engineers, an informal introduction to abstract algebra and elliptic curves is presented. More rigorous treatments can be found in the following books [7, 8, 9, 10].

2.1 Groups and Fields

A group $(G, +)$ consists of a set of numbers G together with a operation $+$ which satisfies the following properties

- Associativity: $(a + b) + c = a + (b + c)$ for all $a, b, c \in G$
- Identity: There is an element $0 \in G$ such that $a + 0 = 0 + a$ for all $a \in G$
- Inverse: For every $a \in G$ there exists an element $-a \in G$ such that $-a + a = a + -a = 0$.

The group G is said to be abelian (or commutative) if

- $a + b = b + a$ for all $a, b \in G$

We will use the notation $\#G$ to denote the number of elements in the group.

A field $(F, +, \times)$ is a set of numbers F together with two operations, $+$ and $\times$ which satisfies the following properties

- $(F, +)$ is an abelian group with identity 0
- $\times$ is associative
- there exists an identity $1 \in F$ with $1 \neq 0$ such that $1 \times a = a \times 1 = a$ for all $a \in F$
- the operation $\times$ is distributive over $+$ i.e. $a \times (b + c) = (a \times b) + (a \times c)$ and $(b + c) \times a = (b \times a) + (c \times a)$ for all $a, b, c \in F$
- $a \times b = b \times a$ for all $a, b \in F$
- for every $a \neq 0, a \in F$ there exists an element $a^{-1} \in F$ such that $a^{-1} \times a = a \times a^{-1} = 1$.

If the field has a finite set of elements, it is called a finite (or Galois) field. Let F_p be the finite field with p elements. Numbers in the field F_2 can be represented by $\{0, 1\}$. If $p = 2^n$, numbers in F_{2^n} can be represented as n-bit binary numbers.

2.2 F_{2^n} Operations (Normal Basis)

The field F_{2^n} has particular importance in cryptography since it leads to particularly efficient hardware implementations. Elements of the field are represented in terms of a basis. Most implementations either use a polynomial basis or a normal basis. For the implementation described in this paper, a normal basis was chosen since it is believed that it leads to a more efficient hardware implementation.

In a normal basis, an element A can be uniquely represented in the form

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i}$$

where $a_i \in F_2$ and $\beta \in F_{2^n}$.

2.2.1 Addition and Squaring

The addition operation over F_{2^n} is simply a bit-wise exclusive OR (XOR) operation. Furthermore, in a normal basis, squaring is simply a rotate left operation.

2.2.2 Multiplication

Let

$$A = \sum_{i=0}^{n-1} a_i \beta^{2^i},$$

$$B = \sum_{i=0}^{n-1} b_i \beta^{2^i}$$

and

$$C = A \times B = \sum_{i=0}^{n-1} c_i \beta^{2^i}$$

then multiplication is defined in terms of a multiplication table $\lambda_{ij} \in \{0, 1\}$

$$c_k = \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} \lambda_{ij} a_{i+k} b_{j+k} \qquad (1)$$

An optimal normal basis (ONB) [11] is one with the minimum number of terms in Equation 1, or equivalently, the minimum possible number of nonzero λ_{ij}. This value is $2n - 1$ and since it allows multiplication with minimum complexity, such a basis would normally lead to a more efficient hardware implementation.

Derivation of the λ_{ij} values in Equation 1 is dependent on n. There exists an optimal normal basis in F_{2^n} if

1. 2 is a primitive in F_{n+1}, or
2. 2 is a primitive in F_{2n+1}, or
3. n is odd and 2 generates the quadratic residues in Z_{2n+1}.

An ONB exists in F_{2^n} for 23% of all possible values of $n < 1000$ [11]. Table 1 lists these values.

The multiplication table is derived differently for the three different types of ONB described above. As an example, for the second type of ONB, $\lambda_{ij} = 1$ iff i and j satisfy one of the four congruence $2^i \pm 2^j \equiv \pm 1$ [11].

The design presented in this paper assumes an n which has an ONB.

2	3	4	5	6	9	10	11
12	14	18	23	26	28	29	30
33	35	36	39	41	50	51	52
53	58	60	65	66	69	74	81
82	83	86	89	90	95	98	99
100	105	106	113	119	130	131	134
135	138	146	148	155	158	162	172
173	174	178	179	180	183	186	189
191	194	196	209	210	221	226	230
231	233	239	243	245	251	254	261
268	270	273	278	281	292	293	299
303	306	309	316	323	326	329	330
338	346	348	350	354	359	371	372
375	378	386	388	393	398	410	411
413	414	418	419	420	426	429	431
438	441	442	443	453	460	466	470
473	483	490	491	495	508	509	515
519	522	530	531	540	543	545	546
554	556	558	561	562	575	585	586
593	606	611	612	614	615	618	629
638	639	641	645	650	651	652	653
658	659	660	676	683	686	690	700
708	713	719	723	725	726	741	743
746	749	755	756	761	765	771	772
774	779	783	785	786	791	796	803
809	810	818	820	826	828	831	833
834	846	852	858	866	870	873	876
879	882	891	893	906	911	923	930
933	935	938	939	940	946	950	953
965	974	975	986	989	993	998	

Table 1: Values of n with an optimal normal basis.

2.2.3 Inversion

The algorithm used for inversion is based on the identity

$$a^{-1} = a^{2^n - 2} = (a^{2^{n-1}-1})^2$$

for all $a \neq 0$ in F_{2^n}. Itoh and Tsujii [12] proposed a method which minimizes the number of multiplications (squarings are much cheaper in a normal basis). It is based upon the following identities

$$a^{2^{n-1}-1} = \begin{cases} (a^{2^{\frac{n-1}{2}}-1})^{2^{\frac{n-1}{2}}} a^{2^{\frac{n-1}{2}}-1} & n \text{ odd} \\ a(a^{2^{\frac{n-1}{2}}-1})^2 & n \text{ even} \end{cases}$$

and can be computed using the following algorithm.

INPUT: $k \in F_{2^n}$
OUTPUT: $l = k^{-1}$

1. Set $s \leftarrow \log_2 n - 1$
2. Set $p \leftarrow k$
3. For i from s downto 0
   Set $r \leftarrow$ shift n to right by s bit(s)
   Set $q \leftarrow p$
   Rotate q to left by r bit(s)
   Set $t \leftarrow$ multiply p by q
   If last bit of r is set then
      Rotate t to left by 1-bit
      $p \leftarrow$ multiply t by k
   else
      $p \leftarrow t$
   $s \leftarrow s - 1$
4. Rotate p to left by 1-bit
5. Set $l \leftarrow p$
6. Return l

The total number of multiplies M required to perform an inversion in F_{2^n} using the above algorithm is

$$M(n) = \log_2(n-1) + \nu(n-1) - 1$$

where $\nu(x)$ is the number of nonzero bits in the binary representation of x.

2.3 Elliptic Curves over F_{2^n}

A nonsupersingular elliptic curve E over F_{2^n}, $E(F_{2^n})$ is the set of all solutions to the following equation with coordinates in the algebraic closure of E [7]

$$y^2 + xy = x^3 + a_2x^2 + a_6. \quad (2)$$

where $a_2, a_6 \in F_{2^n}$ and $a_6 \neq 0$. Such an elliptic curve is a finite abelian group [7]. The number of points in this group is denoted by $\#E(F_{2^n})$.

2.3.1 Curve Addition

If $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ are points on the elliptic curve (i.e. satisfy Equation 2) and $P \neq -Q$ then $(x_3, y_3) = R = P + Q$ can be defined geometrically. In the case that $P \neq Q$, a line intersecting the curve at points P and Q must also intersect the curve at a third point $-R = (x_3, -y_3)$. If $P = Q$, the tangent line is used.

Alegebraically, if $P \neq Q$

$$\begin{aligned} \lambda &= \frac{y_1 + y_2}{x_1 + x_2}, \\ x_3 &= \lambda^2 + \lambda + x_1 + x_2 + a_2, \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1. \end{aligned}$$

If $P = Q$

$$\begin{aligned} \lambda &= \frac{y_1}{x_1} + x_1, \\ x_3 &= \lambda^2 + \lambda + a_2, \\ y_3 &= (x_1 + x_3)\lambda + x_3 + y_1. \end{aligned}$$

2.3.2 Curve Multiplication

Multiplication is defined by repeated addition, i.e.

$$\begin{aligned} Q &= kP \\ &= \underbrace{P + P + \ldots + P}_{k \text{ times}} \end{aligned}$$

This can be computed using the following "double and add" algorithm.

INPUT: $P \in E(F_{2^n})$ and $k \in F_{2^n}$
OUTPUT: kP

1. $k = \sum_{i=0}^{m} b_i 2^i, b_i \in 0, 1, b_m = 1$
2. $Q = P$
3. For i from $m-1$ downto 0
   $Q = Q + Q$
   If $b_i = 1$ then
      $Q = Q + P$
4. Return Q

which requires $n + \nu(k) - 2$ point additions where $\nu(k)$ is the number of nonzero bits in the binary representation of k.

2.4 Discrete Logarithm Problem

Elliptic curve cryptography is based on the discrete logarithm problem applied to elliptic curves over a finite field. In particular, for an elliptic curve E it relies on the fact that it is easy to compute

$$Q = kP$$

for $k \in \{1, \ldots, \#G-1\}$ and $P, Q \in E$, however, there is currently no known subexponential time algorithm to compute k given P and Q.

In fact, the discrete logarithm problem can be used to build cryptosystems with any finite abelian group. Indeed, multiplicative groups in a finite field were originally proposed. However, the difficulty of the problem depends on the group, and at present, the problem in elliptic curve groups is orders of magnitude harder than the same problem in a multiplicative group of a finite field. This feature is the main strength of elliptic curve cryptosystems.

2.5 Elliptic Curve Cryptography

The discrete logarithm problem can be used as the basis of various public key cryptographic protocols for key exchange, encryption and digital signatures. It is beyond the scope of this paper to review all of the cryptographic protocols for public key cryptography, but an example of its use in the Diffie–Hellman key exchange is given in this section.

Suppose that Alice and Bob wish to agree on a common key to be used for encryption using a traditional secret key algorithm such as DES, but need to do so over an insecure channel such as the Internet. Then the following Diffie–Hellman procedure can be used with a public elliptic curve E and point $P \in E$.

1. Alice generates a secret random integer $k_A \in 1, \ldots \#G-1$ and sends the point on E $k_A \times P$ to Bob
2. Bob generates a secret random integer $k_B \in 1, \ldots \#G-1$ and sends the point on E $k_B \times P$ to Alice
3. Alice and Bob can both compute the key $k = k_A \times (k_B \times P) = k_B \times (k_A \times P)$

An adversary, Carol eavesdropping on the channel, can only gain the information E, P, $k_A \times P$ and $k_B \times P$. For Carol to be able to compute k, she must solve the discrete logarithm problem and the best known algorithm takes fully exponential time. Alice and Bob, however, need only compute elliptic curve multiplications which are comparatively easy.

3 An Elliptic Curve Processor

A block diagram of the elliptic curve processor is shown in Figure 1. The organization is similar to a traditional microcoded central processing unit (CPU)

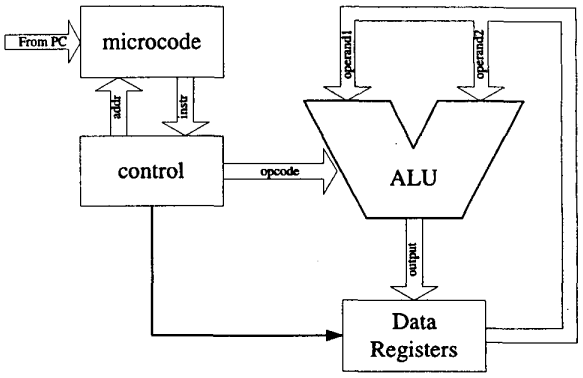

Figure 1: EC processor architecture

that it consists of an arithmetic logic unit (ALU), register file, a microcode sequencer and microcode storage. Major differences between this architecture and a conventional CPU are that the datapath is n bits wide and the ALU performs operations based on F_{2^n} arithmetic instead of integer arithmetic.

3.1 Arithmetic Logic Unit (ALU)

The ALU is faster and simpler than an integer ALU since no carry propagation is required. The complexity is determined by the F_{2^n} multiplier.

Figure 2 shows the circuit used for calculating c_k of Equation 1 [13]. In each cycle t, $(0 \leq t < n)$, the c_k'th cell computes

$$F_k(t) = b_{2k+t} \sum_{i=0}^{n-1} \lambda_{ik} a_{i+k+t}$$

(where all subscripts are reduced modulo n).

In each cycle, the A, B and C registers are rotated as shown in Figure 3. The result being that after n cycles, the contents of register C are the desired product of the A and B inputs [13]. It should be noted that an optimal normal basis reduces the number of interconnections and fanout of signals in the multiplier to the minimum possible, resulting in reduced area and increased speed over a non–optimal normal basis. In fact, the maximum fanout for a_i in Figure 3 is 4 [13].

The addition operation is implemented simply as an XOR function and the squaring function is implemented as a rotate left operation.

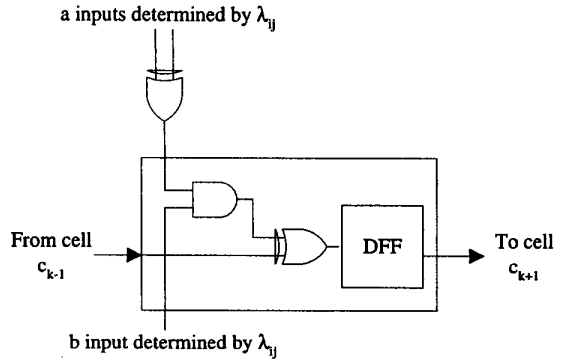

Figure 2: F_{2^n} multiplier element of c_k

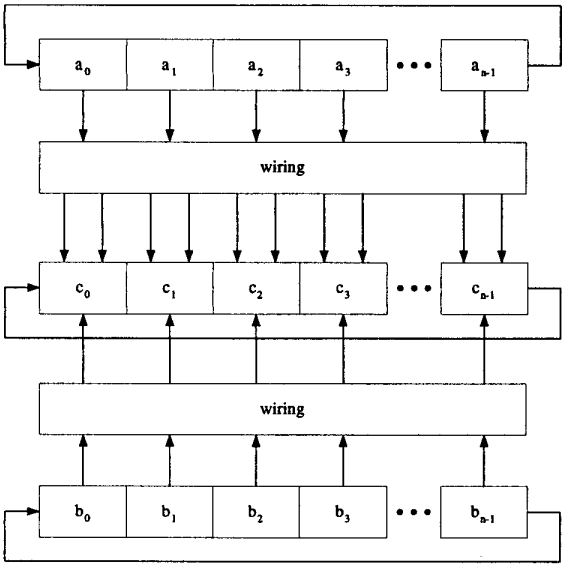

Figure 3: F_{2^n} multiplier circuit

3.2 Register File

A 16 × n-bit dual-port synchronous register file is constructed from the 16 × 1-bit distributed RAM feature of the Xilinx Virtex series logic cell (see Section 3.5). This gives an eight-fold reduction in resources over RAMs based on latches.

3.3 Microcode

The ALU plus register file form a F_{2^n} processor similar to previous designs [3]. However, for performing elliptic curve cryptography, higher level elliptic curve multiplications of Section 2.3.2 are required. This could be implemented as a finite state machine [5] or in microcode. The implementation described in this paper opted for a microcoded approach which has the following advantages in a FPGA implementation

1. the microcode is stored in Xilinx Virtex block RAMs and do not use logic resources of the FPGA (as explained in Section 3.5). The microcode sequencer in this design is very simple and has a small overhead.

2. the microcode can be changed without requiring recompilation of the elliptic curve processor

3. algorithmic optimizations to the processor can be performed entirely in microcode

4. a microcoded description is a higher level abstraction than a finite state machine and hence easier to develop and debug.

The instruction set of the processor is shown in Table 2. Apart from instructions which directly control the ALU, there are three types of jump instructions: JMP – jump unconditionally, JKZ – jump if the least significant bit of K counter is zero and JCZ – jump if the C register is zero.

Each instruction is 16 bits in width and the format of instructions is shown in Figure 4. Most instructions accept a source register and a destination register in `operand1` and `operand2` respectively. The jump instructions have a 12-bit jump address.

A two pass symbolic assembler was developed which takes symbolic input and produces the binary microcode which can be downloaded to the processor's microcode store. A microcode simulator was also written to facilitate microcode development.

Operation	Clock Cycles
NOP	1
XOR	1
Rotate left, ROTL	1
Shift right, SHFR	1
Field Multiplication, MUL	n+1
Transfer register value, TFR	1
Jump Instructions JKZ, JCZ, JMP	1

Table 2: Instruction set

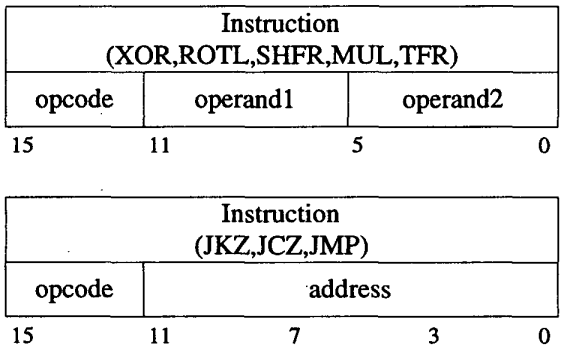

Figure 4: Instruction format

3.4 Parameterized Module Generator

An important feature of the elliptic curve processor (ECP) is that it is parameterized via a module generation program so that elliptic curve processors for any n with an optimal normal basis can be produced. Hence this scheme advantageously uses the reconfigurable nature of the FPGA to add the flexibility of being able to choose arbitrary n (compared with fixed n of previous designs [3]).

The module generator is a program written in the Perl programming language [14] which takes n as an input parameter and produces the VHDL code of the ECP as output. Perl is a language which supports long integer arithmetic which was helpful in performing the arithmetic required to generate the λ matrix of the field multiplier.

3.5 Implementation

The ECP was implemented on an Annapolis Micro Systems Wildcard^{TM} board [15], a type II PCM-CIA card with 32-bit CardBus interface consisting of a Processing Element (PE, Xilinx Virtex XCV300–4 [16]) and two 64K by 32-bit SDRAMs. The XCV300 has 64Kbits of block RAM (arranged as $8 \times 8 - Kbit$ blocks) and 1536 configurable logic blocks (CLBs).

The basic building block of the Virtex FPGA is the the logic cell (LC). A LC includes a 4-input function generator, carry logic and a storage element. Each Virtex CLB contains four LCs, organized in two slices. The 4-input function generator are implemented as 4-input look-up tables (LUTs). Each of them can provide the functions of one 4-input LUT or a 16×1-bit synchronous RAM (called "distributed RAM"). Furthermore, two LUTs in a slice can be combined to create a 16×2-bit or 32×1-bit synchronous RAM, or a 16×1-bit dual-port synchronous RAM.

Also within the Virtex chip are dedicated blocks of dual-ported 8-Kbit block RAMs, configurable in widths from 1 to 16-bits. These were used in our design for the storage of microcode and for communications between the host and the PE.

4 Results

VHDL code for the elliptic curve processor was generated for different values of n which have an optimal normal basis. The code was then synthesized using Synposys FPGA Express and the Xilinx Foundation Tools. Table 3 shows the resource utilization and maximum clock rate reported by the Xilinx tools for designs with different n.

The size of the microcode is currently less than 512 16-bit words and does not significantly change for different n.

n	# of slices	Max. freq (MHz)
113	1290	45
155	1567	36
281	2622	33

Table 3: Resource utilization and maximum clock rate for different n on a Xilinx XCV300–4. The Xilinx XCV300 contains 3072 slices (1536 CLBs).

The total number of cycles required for an elliptic curve multiplication for various n is given in Table 4, where we assume that the k of Equation 3 is a n-bit binary number with 64 bits set. The time required for an elliptic curve multiplication at the maximum

frequency is shown in Table 4. Note that these figures do not include the time for host processor interfacing. In many applications, the P of Equation 3 only needs to be downloaded once to the processor. It may also be possible to generate k on the chip to enjoy further savings on the I/O bandwidth of the chip.

The execution time of the processor was compared with that of an optimized software implementation of an optimal normal basis elliptic curve package [9] running on a 270MHz Sun Ultra-5 with 512MB of RAM. The results are presented in Table 4. The software package uses many optimizations (detailed in [9]) currently not implemented in our microcode such as a balanced integer representation [17] and faster inversion [18]. It can be seen that the elliptic processor is approximately 30 faster than the software implementation. However, it should be noted that for many embedded applications, compact implementation, low cost and low power consumption are often of greater concern that maximum speed.

n	# of cycles	SW time Ultra-5(ms)	HW time (ms)	Speed-up
113	166783	110	3.7	30
155	246443	180	6.8	27
281	474504	516	14.4	36

Table 4: Execution time for elliptic curve multiplication and comparison with a software implementation.

The EC processor was successfully tested on the Wildcard board, the microcode instructions being downloaded to the Block RAMs by the host PC. Dual-ported distributed RAM was used to interface the processor to the host.

5 Conclusion

A processor which can perform elliptic curve operations was presented. The datapath of the processor is an optimal normal basis F_{2^n} processor synthesized by a parameterized module generator which can accommodate arbitrary n. The control part of the processor is microcoded to facilitate rapid development and algorithmic optimizations. The design was successfully tested on a Xilinx Virtex XCV300-4 device and could perform an elliptic curve multiplication over the field $F_{2^{113}}$ in 3.7 milliseconds which is thirty-fold speedup over an optimized software implementation.

6 Further work

There has been much recent work on improved algorithms for implementing the field and curve operations required for elliptic curve cryptography [9, 17, 19, 20]. We are currently implementing some of these ideas and exploring the tradeoffs between affine and projective coordinates [7]. These improvements are at the algorithmic level and can be implemented completely in microcode. It is expected that they will lead to large improvements in performance.

References

[1] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, pp. 203–209, 1987.

[2] V. S. Miller, "Use of elliptic curves in cryptography," in *CRYPTO '85*, pp. 417–426, 1986.

[3] G. B. Agnew, R. C. Mullin, and S. A. Vanstone, "An implementation of elliptic curve cryptosystems over $F_{2^{155}}$," *IEEE Transactions on Selected Areas in Communications*, vol. 11, pp. 804–813, 1993.

[4] J. S. P. Ivey, S. Walker and S. Davidson, "An ultra–high speed public key encryption processor," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, pp. 19.6.1–19.6.4, 1992.

[5] M. Rosner, *Elliptic Curve Cryptosystems on reconfigurable hardware.* Master's Thesis, Worcester Polytechnic Institute, Worcester, USA, 1998.

[6] G. Orlando and C. Paar, "A super-serial Galois field multiplier for FPGAs and its application to public key algorithms," in *Proceedings of the IEEE Symposium on Field-programmable custom computing machines (FCCM'99)*, pp. 232–239, 1999.

[7] A. J. Menezes, *Elliptic Curve Public Key Cryptosystems.* Kluwer Academic Publishers, 1993.

[8] A. J. Menezes, P. van Oorschot, and S. A. Vanstone, *Handbook of Applied Cryptography.* CRC Press, 1999.

[9] M. Rosing, *Implementing Elliptic Curve Cryptography.* Manning, 1998.

[10] I. Blake, G. Seroussi, and N. Smart, *Elliptic Curves in Cryptography.* Cambridge University Press, 1999.

[11] R. C. Mullin, I. M. Onyszchuk, S. A. Vanstone, and R. M. Wilson, "Optimal normal bases in $GF(p^n)$," *Discrete Applied Mathematics*, vol. 22, pp. 149–161, 1988/89.

[12] T. Itoh and S. Tsujii, "A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases," *Info. and Comput.*, vol. 78(3), pp. 171–177, 1988.

[13] G. B. Agnew, R. C. Mullin, I. M. Onyszchuk, and S. A. Vanstone, "An implementation for a fast public-key cryptosystem," *Journal of Cryptology*, vol. 3, pp. 63–79, 1991.

[14] L. Wall, T. Christianson, and R. L. Schwartz, *Programming Perl.* O'Reilly, 2nd ed., 1996.

[15] Annapolis Micro Systems, Inc., *Wildcard Reference Manual*, 1999. Revision 1.1.

[16] Xilinx, *Virtex 2.5V field programmable gate arrays*, 1999.

[17] J. A. Solinas, "An improved algorithm for arithmetic in a family of elliptic curves," in *CRYPTO '97*, pp. 357–371, 1997.

[18] R. Schroeppel, H. Orman, S. O'Mally, and O. Spatscheck, "Fast key exchange with elliptic curve systems," in *CRYPTO '95*, pp. 43–56, 1995.

[19] J. Guajardo and C. Paar, "Efficient algorithms for elliptic curve cryptosystems," in *CRYPTO '97*, pp. 343–356, 1997.

[20] J. Lopez and R. Dahab, "An improvement of the Guajardo-Paar method for multiplication on non-supersingular elliptic curves," in *Proceedings of the XVIII International Conferences of the Chilean Society of Computer Science (SCCC'98)*, pp. 91–95, 1998.