Elliptic Curve Scalar Multiplier Design Using FPGAs

Lijun Gao, Sarvesh Shrivastava, and Gerald E. Sobelman

Dept. of Electrical & Computer Engineering, University of Minnesota, Twin-Cities
200 Union Street S.E., Minneappolis, MN 55455, U.S.A
Email: {lgao,sobelman}@ece.umn.edu
URL: http://www.ece.umn.edu/users/lgao

Abstract. A compact fast elliptic curve scalar multiplier with variable key size is implemented as a coprocessor with a Xilinx FPGA. This implementation utilizes the internal SRAM/registers of the FPGA and has the whole scalar multiplier implemented within a single FPGA chip. The compact design helps reduce the overhead and limitations associated with data transfer between FPGA and host, and thus leads to high performance. The experimental data from the mappings over small fields shows that the carefully constructed hardware architecture is regular and has high CLB utilization.

Keywords. elliptic curves, public-key cryptography, scalar multiplication, Galois field, reconfigurable hardware, FPGA, coprocessor

1 Introduction

The motivation of this work is to develop high-speed elliptic curve scalar multipliers with the least development time, the lowest hardware cost and maximal flexibility. Recently, elliptic curve (EC) cryptosystems have become attractive due to their small key sizes and varieties of choices of the curves available. Their low cost and compact size are critical to some applications, including smart cards and hand-held devices[1]. In all those applications, an elliptic curve scalar multiplier serves as a basic building block for secret key exchange, authentication and certification. Most microprocessors have hardware-supported integer multiplication and logic functions like AND, OR or XOR, so an elliptic curve cryptosystem can be implemented on them. However, this is not efficient because of word size mismatch, less parallel computation, no hardware supported wire permutation and algorithm/architecture mismatch. As a result, such systems have low performance/cost ratios.

The solution to this problem is to build a coprocessor as a dedicated computing unit. Moreover, using an FPGA, the coprocessor can be reconfigured for different application instances or for different computation stages of one particular application. Thus, the total hardware utilization can be kept at a very high rate and the computation is speeded up.

Previous work in this area is based on a coprocessor for arithmetic operations over $GF(2^{155})$ using a gate array[11]. To accomplish an elliptic curve operation,

Ç.K. Koç and C. Paar (Eds.): CHES'99, LNCS 1717, pp. 257–268, 1999.

the host controller and the coprocessor have to transfer data between each other frequently. The control of elliptic curve operations and the storage of intermediate variables are provided by the host controller. Therefore, the communication cost is large and may be a bottleneck for an elliptic curve cryptosystem.

In this paper, a compact fast elliptic curve scalar multiplier coprocessor is introduced which utilizes the internal SRAM/registers in an FPGA and is implemented within a single FPGA chip. The normal bases for the underlying finite field are chosen because the field squarings can be done with the bit shifts in hardware and are virtually free[2]. A pipelined digit-serial modified Massey-Omura multiplier is constructed and is used in the design. The scalar multiplier is implemented with a parameterized (in term of key size) VHDL description and is synthesized/mapped to a Xilinx FPGA. By changing the parameter for key size and re-doing synthesis, a different instance can be acquired. The architecture and algorithms adopted here are suitable for massively parallel computation. Therefore, with larger capacity FPGA chips, higher performance can be easily obtained with few changes in the underlying design.

2 Algorithm of EC Scalar Multiplier

The basic operation in an EC cryptosystem is the scalar multiplication over the elliptic curve and the most efficient method for computing EC scalar multiplications is to use an addition/subtraction method[2][4][5]. With this method, the scalar (or the integer) is decomposed in a non-adjacent format(NAF) and one scalar multiplication is done with a series of additions/subtractions of elliptic curve points. In turn, each addition/subtraction of EC points consists of a series of underlying field additions, squarings, multiplications and inversions. When the elliptic curve is defined over $GF(2^m)$ with an optimal normal basis, these underlying field operations have the least complexity. The elliptic curve used in this implementation is defined by Weierstrass equations as:

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

where $a, b \in GF(2^m)$ and $b \neq 0$. The algorithms of EC scalar multiplication and EC addition/subtraction are shown below respectively[2].

Algorithm 1: EC scalar multiplication
Input:

$P-$EC point
$n-(e_{l-1}, e_{l-2}, ..., e_1, e_0)$ non-adjacent format integer and e_{l-1}=1

Output:

$Q = nP$

Computation:

1. $Q = P$;
2. For $i = l - 2$ downto 0 do

Set $Q = 2Q$;
If $e_i = 1$ then set $Q = Q + P$;
If $e_i = -1$ then set $Q = Q - P$;
3. Output Q;

Algorithm 2: EC addition
Input:
$P_0 = (x_0, y_0)$
$P_1 = (x_1, y_1)$
Output:
$(x_2, y_2) = P_2 = P_1 + P_0$
Computation:
1. If $P_0 = 0$, then output $P_2 = P_1$ and stop;
2. If $P_1 = 0$, then output $P_2 = P_0$ and stop;
3. If $x_0 = x_1$ then
If $y_0 = y_1$ then
$\lambda = x_1 + y_1/x_1$;
$x_2 = \lambda^2 + \lambda + a$;
$y_2 = x_1^2 + (\lambda + 1)x_2$;
else output O;
else
$\lambda = (y_0 + y_1)/(x_0 + x_1)$;
$x_2 = \lambda^2 + \lambda + x_0 + x_1 + a$;
$y_2 = (x_1 + x_2)\lambda + x_2 + y_1$;
4. Output(x_2, y_2);

Since $-P_0 = (x_0, x_0 + y_0)$ for $P_0 = (x_0, y_0)$ and $P_1 - P_0 = P_1 + (-P_0)$, EC subtraction is as simple as EC addition and can be computed with one EC addition. The average and maximal number of non-zero bits among NAFs are about $m/3$ and $m/2$, respectively[2]. Therefore the average cost of Alg.1 is about m point doubles and $m/3$ point additions/subtractions[2], and the worst case cost is about m point doubles and $m/2$ point additions/subtractions. This is much better than binary decomposition, which has $m/2$ non-zero bits on average and m non-zero bits in the worst case.

3 Hardware Architecture

3.1 System Structure

Since the word size (or key size) for a typical elliptic curve cryptosystem is large, the above algorithm can not be unfolded. Therefore, a folded hardware architecture is constructed with a controller to sequence the computation. The underlying field multiplier, a $GF(2^m)$ multiplier with optimal normal basis, can be implemented as either a serial multiplier or a digital-serial multiplier or a parallel multiplier, depending on the amount of available hardware resources. In

Fig. 1, the two FIFOs serve as input/output buffers and the dual-port register file is used to save input parameters and intermediate data. This is realizable because Xilinx FPGAs have a large amount of internal SRAM and registers. Alternatively, if we were to use external SRAM, it would take either many external user pins with multiple SRAM chips, or multiple cycles to read in one single word. This would result in low bandwidth data transfers due to the large word size. The hardware provides $GF(2^m)$ arithmetic units GF_adder, GF_squarer, GF_multiplier and GF_inverter. With the finite field of characteristic 2 as the underlying field, addition is just a bit-wise XOR, and with normal bases representation, squaring is a simple cyclic right shift. The internal structures of the GF_multiplier and GF_inverter are given in the following sections.

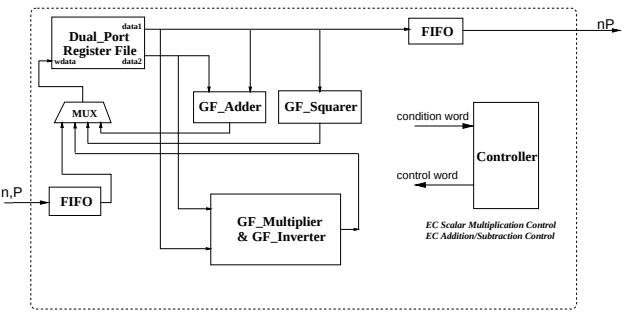

Fig. 1. Hardware architecture of EC coprocessor

3.2 GF_multiplier Structure

The structure of GF_multiplier is a modified form of the Massey-Omura multiplier. Compared to the implementation in [9], the modified structure reduces the number of AND gates and the wire permutation by 50% in the AND

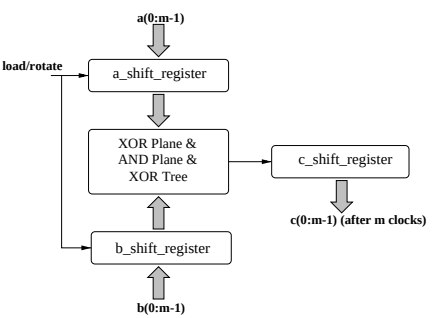

Fig. 2. Structure of $GF(2^m)$ serial multiplier

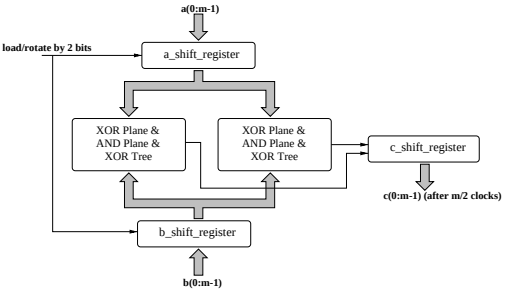

Fig. 3. Structure of $GF(2^m)$ digit-serial multiplier

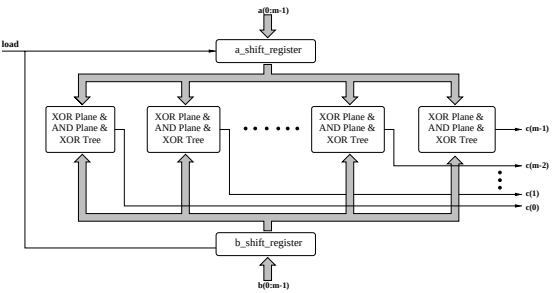

Fig. 4. Structure of $GF(2^m)$ parallel multiplier

Plane without changing the total number of XOR gates. A serial multiplier of such a structure is shown in Fig. 2, which can be simply unfolded to a digital-serial multiplier in Fig. 3 or a parallel multiplier in Fig. 4. In this serial multiplier, at each cycle, a_shift_register and b_shift_register make a cyclic right shift, and one bit of the product is computed and shifted into the product register, c_shift_register. Therefore, each multiplication takes m clock cycles. If the serial multiplier is unfolded to a parallel multiplier, then each multiplication only takes one clock cycle. However, the required hardware will be m times that of the serial multiplier. Pipeline techniques are also applied to the XOR Plane—AND Plane—XOR Tree of the multiplier to reduce the clock cycle time. The modified Massey-Omura serial multiplier takes m AND gates, $2m$ XOR gates and $3m$ flip-flops, and has a latency of $m \times (T_{AND} + T_{XOR}\lceil log_2(m-1)\rceil)$ when it is not pipelined. However when it is pipelined, the serial multiplier has a latency of $(m + \lceil log_2(m-1)\rceil) \times Max(T_{AND}, T_{XOR})$ and a total cost of m AND gates, $2m$ XOR gates and $5m$ flip-flops. It is obvious that the pipelined multiplier is much faster than non-pipelined ones when m is large. The same techniques can also apply to a digit-serial multiplier. The digit-serial multiplier with a digit size of k will generate k bits of the product simutaneously and thus one multiplication can be done in kth fold time taken by a serial multiplier. The digit-serial multiplier makes a trade-off of the speed and the hardware between a serial multiplier and a parallel multiplier.

3.3 GF_inverter Structure

The structure of GF_inverter is derived from the method introduced by T. Itoh et al[8] and is shown in Fig. 5. The inverse takes $\lfloor log_2(m-1) \rfloor$ recursive iterations and a total of $\lfloor log_2(m-1) \rfloor + HW(m-1) - 2$ underlying field multiplications, where $HW(m-1)$ is the Hamming weight of $(m-1)$.

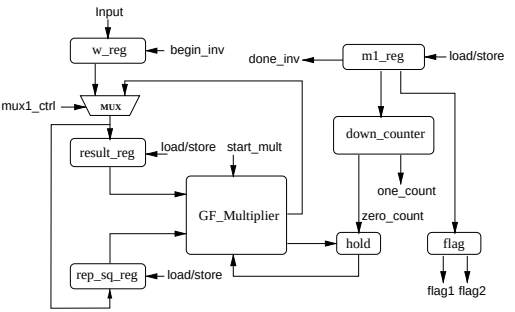

Fig. 5. Structure of $GF(2^m)$ inverter

3.4 Controller Structure

The controller takes advantage of the abundance of internal SRAM and registers in Xilinx FPGAs. The controller is built up as a finite state machine with table look-up to implement the logic functions. Since the whole look-up table consists of small look-up tables from each CLB (configurable logic block), the controller can be pipelined to have a clock cycle time equal to one CLB delay. A pipelined structure of the controller is illustrated in Fig. 4. At each cycle, a selector associated with the value of PC is generated. It then selects the appropriate bit of the condition word to make PC either increment or load a new value from the Branch PC look-up. In case of a branch hazard, the pipeline register is cleared.

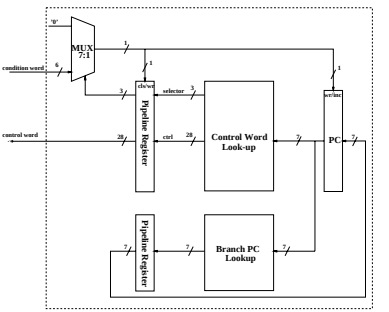

Fig. 6. Pipelined structure of the controller

4 Dataflow of EC Scalar Multiplication

The algorithm and hardware architecture leads to the computation dataflow chart shown in Fig. 7 and Fig. 8. Each bit of the decomposed scalar is encoded by 2 bits to represent {1,0,-1}. A total of 9 intermediate storage elements are needed and it leads to a 4-bit address space for the dual-port register file. Then each data bit of the dual-port register file can be implemented with one CLB and the dual-port register file has one CLB delay. It is obvious that the dataflow has many conditional branches. Therefore, the branch hazard problem has to be taken care. From the dataflow chart, the schedule and control of computation in the scalar multiplier is worked out and the corresponding VHDL description is implemented.

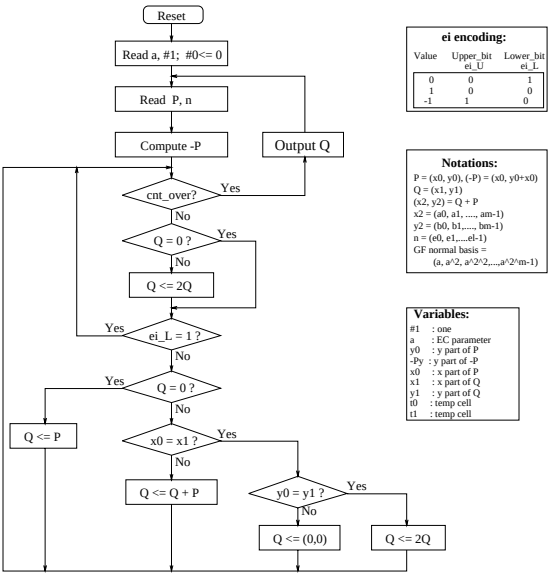

Fig. 7. Dataflow chart I of EC scalar multiplier

5 Development of EC Scalar Multiplier

All components in the scalar multiplier are implemented according to above algorithms and hardware architectures. A pipelined digit-serial multiplier is implemented because a serial or a parallel multiplier is only a special case for digit size of 1 or m respectively. The controller is a finite state machine according to Fig. 6. The implementation of the controller follows the dataflow charts I & II in Fig. 7 and Fig. 8, respectively. All operations in the dataflow can be categorized as one of the following atomic operations: unconditional jump, conditional

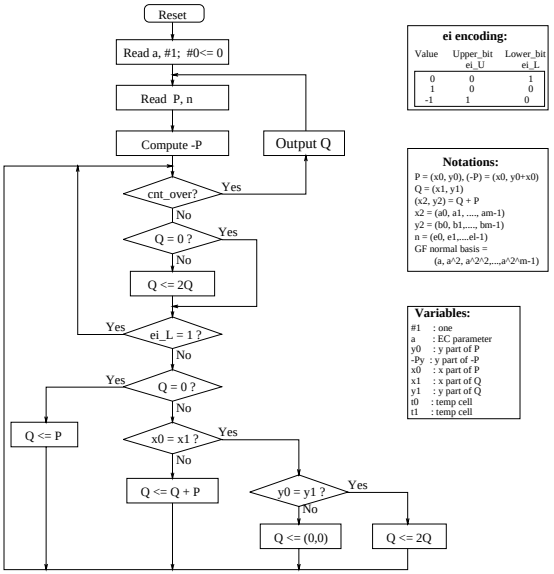

Fig. 8. Dataflow chart II of EC scalar multiplier

jump, operand load, operand store, finite field addition, finite field squaring, finite field multiplication and finite field inversion. Then, each state of the controller consists of one or more such atomic operations because addition, squaring, multiplication/inversion and load/store can be executed concurrently. The execution schedule is optimized to provide the shortest computing time. These atomic operations are represented as macros and are re-used in the VHDL code. One example of the VHDL simulation results is shown in Fig. 9 for $m = 39$ and type II optimal normal basis[14]. The example shows two scalar multiplications which compute

$$7 \times (1A0C3EB323, 2EE60CF558) = (7527E64FAD, 34A9265CF1)$$

$$7 \times (1A0C3EB323, 34EA32467B) = (7527E64FAD, 418EC0135C)$$

for $a = 1A28CE01DD$ and $b = 1200569A44$ in equation (1). The hex encoding follows the method in [14].

6 Mapping onto Xilinx XC4000XL-Series FPGA

6.1 Synthesis/Mapping Results

After VHDL code simulations, the design is setup with a pipelined $GF(2^m)$ serial multiplier (or digit size of 1) and is mapped onto a Xilinx XC4000XL-series FPGA for some small values of m. The synthesis is done with Exemplar synthesis tools and the mapping/layout is done with Xilinx Design Manager.

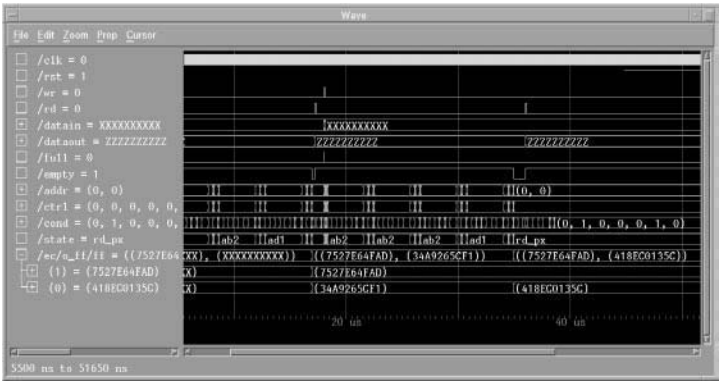

Fig. 9. VHDL simulation waveform for $m = 39$

The results are shown in Table 1 and one example of the layout for $m = 29$ is shown in Fig. 10. The mapping efficiency is represented with the percentage of total CLBs in a FPGA that is used by the design and the throughput is represented with the scalar multiplications per second. Both the throughput and clock cycles in Table 1 represent the worst case performance. It is shown that the architecture of the prototype is regular and the designed coprocessor has very high CLB utilization. The dominant operation in the EC scalar multiplier is $GF(2^m)$ multiplication. Therefore, if the $GF(2^m)$ serial multiplier is unfolded with a factor of 2, then the throughput, in terms of scalar multiplications per second, will be doubled.

Table 1. FPGA chip area utilization and throughput

Value of m	XC4000XL Device	CLB Usage	Clock Cycles	Throughput (scalar mul/sec)
5	4010XL	272/400 = 68%	126	179856
11	4013XL	478/576 = 83%	825	19230
29	4028XL	962/1024 = 93%	7158	1653
53	4044XL	1626/1936 = 84%	26753	417

6.2 Analysis of the Mapping Results

Since the proposed hardware architecture is regular and simple, the expected mapping results can also be obtained with an estimation formula. Then, a comparison can be made to analyze the mapping results. In order to derive the formula, two summaries are listed below:

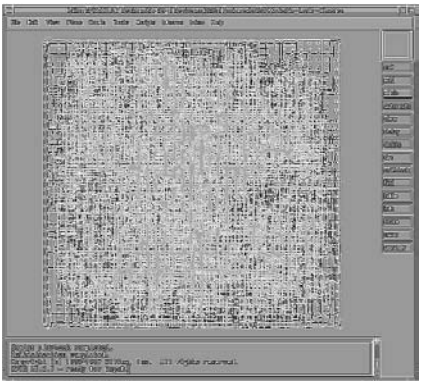

Fig. 10. FPGA Layout for m=29 with XC4028XL

1. CLB (Configurable Logic Block) structure of Xilinx XC4000XL-series:
   - 2 flip-flops(FFs) per CLB
   - 2 function generators (FGs) per CLB (4 input/single output logic unit)
   - 2 single-port 16x1 RAMs per CLB (using two logic units)
   - 1 dual-port 16x1 RAM per CLB (using two logic units)
2. Cost of implementing basic components:
   - two m-bit registers take $2m$ FFs.
   - three m-bit shift registers take $3m$ FFs and FGs.
   - four m-bit 2:1 MUXs take $4m$ FGs.
   - one m-bit GF_Adder takes m FGs.
   - one m-bit dual-port 9 word RAM takes $2m$ FGs.
   - one m-bit 6 word FIFO takes $6(m+1)$ FFs and FGs.
   - one m-bit 2 word FIFO takes $2(m+1)$ FFs and FGs.
   - one m-bit GF_Multiplier takes $5m$FFs and $3m$ FGs (pipelined bit-serial multiplier).
   - one m-bit GF_Inverter takes $3(m + log_2 m)$FFs and FGs (excluding GF_Multiplier).

From above basic facts, the cost of one EC scalar multiplier with key size m is derived as:

- Total FFs $= 21m + 3log_2 m + 48$
- Total FGs $= 24m + 3log_2 m + 308$
- Minimal value of Total CLBs
  $=$ max(Total FFs, Total FGs)$/2$
  $= 12m + (3log_2 m)/2 + 154$
- Maximal value of Total CLBs
  $=$ (Total FFs + Total FGs)
  $= 45m + 6log_2 m + 356$

Table 2 is constructed with above estimation formula and the data in Table 1. In Table 2, the actual CLBs means the CLB count of the mapping obtained

with Xilinx Design Manager. The Min/Max CLBs comes from the estimation formulas and puts a lower/upper limit for the total number of CLBs needed for the design. Using regression, a polynomial curve of degree 2 is fitted with the data of the actual CLBs and the CLB usage for larger fields are predicted through extrapolation with the fitted curve. However routing resources are not taken into account in the extrapolation and they could become a bottleneck for larger fields. For m=160, the estimated number of CLBs (≈ 4000) is larger than the capacity of the XC4085XL. However, the design should easily fit onto the Virtex XCV1000 chip.

Value of m	Expected Device	Actual CLBs	Min CLBs	Max CLBs
5	XC4010XL	272	219	600
11	XC4013XL	478	292	876
29	XC4028XL	962	510	1692
53	XC4044XL	1626	799	2778

Table 2. Mapping analysis

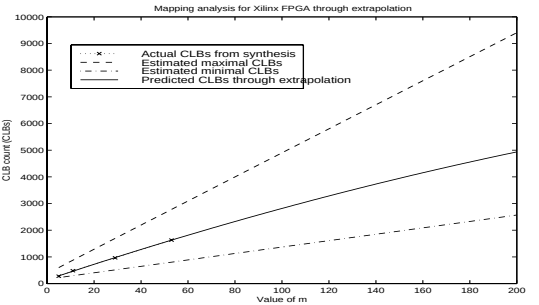

Fig. 11. Extrapolated mapping analysis

7 Conclusions and Future Work

The experimental results from the mappings over small fields show that the hardware architecture is regular and achieves high CLB utilization and high speed. The use of an FPGA in the development of an elliptic curve scalar multiplier demonstrates many advantages:

- Reduced development time and cost.
- Tailorable design for a particular application.
- Reduced hardware overhead and high performance.
- Increased chip area utilization.
- Hardware performance with advantages of software development.
- Simplified hardware architecture and ability to easily add new functions.

The effectiveness and eventual performance/cost ratio of applying reconfigurable hardware to cryptography depends on many factors and research in this area is highly experimental. Therefore, future work remains in many areas and is summarized as follows:

- To map the design onto more types of FPGA chips to show the usefulness of the design and to reveal the relationship between the algorithm and the architecture/resources of FPGAs.

– To build some EC application systems, such as an EC digital signature or an EC Diffie-Hellman key exchange[1][14][15], by using reconfigurable hardware, such that a direct comparison can be made with other implementations using microprocessors[12][13].

Acknowledgments

The authors would like to thank Professor Keshab Parhi for valuable conversations. This research was supported by Defense Advanced Research Project Agency under contract number DA/DABT63-96-C-0050. The authors are grateful to the anonymous reviewers for their constructive comments and suggestions.

References

1. A. J. Menezes, *"Elliptic curves and cryptography",* Dr. Dobb's Journal, April, 1997.
2. J. A. Solinas, *"An Improved algorithm for arithmetic on a family of elliptic curves",* Journal of Cryptography, 1997.
3. N. Kobliz, *"CM curves with good cryptographic properties",* Proc, Crypto '91, Springer-Verlag, 1992, pp.279-287.
4. F. Morain and J. Olivos, *"Speeding up the computations on an elliptic curve using addition-subtraction chains",* Information Theory Application 24(1990), pp.531-543.
5. D. M. Gordon, *"A survey of fast exponentiation methods",* Journal of Algorithms 27, 129-146, 1998.
6. G. B. Agnew, t. Beth, R. C. Mullin and S. A. Vanstone, *"Arithmetic operations in $GF(2^m)$",* Journal of Cryptology, Vol 6, pp.3-13, 1993.
7. D. W. Ash, I. F. Blake and S. A. Vanstone, *"Low complexity normal bases",* Discrete Applied Mathematics 25, 1989, pp.191-210.
8. T. Itoh and S. Tsujii, *"A fast algorithm for computing multiplicative inverses in $GF(2^m)$ using normal bases",* J. Society for Electronic Communications(Japan)44(1986), 31-36.
9. C. C. Wang, T. K. Truong, H.M.Shao, L.J.Decutsch, J.K.Omura and I.S.Reed, *"VLSI architectures for computing multiplications and inverses in $GF(2^m)$",* IEEE Tran. Comp., Vol c-34, No 8, August 1985.
10. 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.
11. G. B. Agnew, R. C. Mullin and S. A. Vanstone, *"An implementation of elliptic curve cryptosystems over $F_{2^{155}}$",* Journal of Cryptology, Vol 3, pp.63-79, 1991.
12. M. J. Wiener, *"Performance comparison of public-key cryptosystems",* RSA Laboratories's CryptoBytes, Vol. 4, No. 1 - Summer 1998.
13. H. Handschuh and P. Paillier, *"Smart card crypto-coprocessor for public-key cryptography",* RSA Laboratories's CryptoBytes, Vol. 4, No. 1 - Summer 1998.
14. American Bankers Association, *X9.62-199x,Public Key Cryptography For The Financial Services Industry: The Elliptic curve digital signature algorithm (ECDSA),* Working Draft, Nov. 1997
15. IEEE, *"IEEE P1363 standard specification for public key cryptography",* draft, 1998.
16. Xilinx, *"The Programmable logic data book",* Xilinx, Inc., 1998.