An Adaptable And Scalable Asymmetric Cryptographic Processor

Neil Smyth, Máire McLoone and John V. McCanny
The Institute of Electronics, Communications and Information Technology (ECIT)
Queen's University of Belfast, Belfast, Northern Ireland
Email: (neil.smyth@conexant.com, maire.mcloone@ecit.qub.ac.uk, john.mccanny@ecit.qub.ac.uk

***Abstract*— In this paper a novel scalable public-key processor architecture is presented that supports modular exponentiation and Elliptic Curve Cryptography over both prime *GF(p)* and binary $GF(2^n)$ extension fields. This is achieved by a high performance instruction set that provides a comprehensive range of integer and polynomial basis field arithmetic. The instruction set and associated hardware are generic in nature and do not specifically support any cryptographic algorithms or protocols. Firmware within the device is used to efficiently implement complex and data intensive arithmetic. A firmware library has been developed in order to demonstrate support for numerous exponentiation and ECC approaches, such as different coordinate systems and integer recoding methods. The processor has been developed as a high-performance asymmetric cryptography platform in the form of a scalable Verilog RTL core. Various features of the processor may be scaled, such as the pipeline width and local memory subsystem, in order to suit area, speed and power requirements. The processor is evaluated and compares favourably with previous work in terms of performance while offering an unparalleled degree of flexibility.**

I. Introduction

Exponentiation is a fundamental cryptographic operation in asymmetric cryptography. However, the increasingly large key lengths that are required to maintain sufficiently high levels of security impose increasing demands on the performance and memory requirements of both hardware and software solutions.

Elliptic Curve Cryptography (ECC) was independently proposed by Koblitz [1] and Miller [2] in 1985. ECC is being adopted as an alternative to exponentiation by an increasing number of standards due to a number of possible advantages. These include lower memory requirements and key lengths for any given security strength and lower complexity arithmetic.

A range of mathematical methods may be applied to both exponentiation and ECC in order to vary the performance, memory and power requirements. Such mathematical "tricks" include Montgomery reduction [3], Barrett reduction [4] and integer recoding [5] schemes such as *m-ary*, *non-adjacent form (NAF)* and windowed *NAF* (*w-NAF*). This paper details a processor with a highly efficient mathematical instruction set suitable for public-key cryptography. This is an extension of previous work [6] in which a programmable exponentiation processor was described. The architecture has been revised and extended to accelerate Montgomery reduction/multiplication and Montgomery inversion [7], to incorporate a range of polynomial arithmetic instructions and provide a scalable architecture. The resulting processor offers an instruction set that is sufficiently generic to support a wide range of applications and mathematical "tricks".

ECC over binary fields may be supported by either polynomial or normal basis arithmetic involving large numbers. For the purposes of this work the seemingly more popular polynomial basis implementations are provided by a range of polynomial instructions. Exponentiation and ECC over prime fields are both enabled by the integer field arithmetic instructions.

The implementation described in this paper differs from previous implementations in the following ways.

1) The processor is an entirely self-contained platform for public-key cryptography, capable of all necessary computation such that external logic is required only to transfer the necessary data to and from the processor. All operations may be performed within the processor thus reducing both memory bandwidth within a SoC system and the required MIPS of any host microprocessor.
2) Variables are stored in local RAM rather than registers providing the ability to operate arbitrary length fields.
3) Integer recoding is provided by software with hardware support to provide resistance to side-channel attacks.
4) A highly efficient arithmetic instruction set allows complex mathematical algorithms to be abstracted and provided by a relatively small number of instructions.
5) The power requirements, performance and resource requirements may also be scaled when the system is synthesised by varying the local memory bandwidth, processor pipeline width and instruction set.

In the next section a brief background on ECC and exponentiation is provided. Section III presents a description of the novel scalable processor public-key architecture, while Section IV provides an evaluation of the processor performance. Exponentiation is demonstrated by a range of integer recoding schemes used with appropriately modified *square-and-multiply* algorithms. ECC scalar point multiplications in both prime and binary extensions fields, *GF(p)* and $GF(2^n)$, are demonstrated using both integer recoding with appropriately modified *double-and add* algorithms and with a range of coordinate systems. For both prime and binary ECC curves and exponentiation the performance for a variety of field lengths are presented. Finally, the merits of the presented architecture

Application-specific Systems, Architectures and Processors (ASAP'06)
0-7695-2682-9/06 $20.00 © 2006 IEEE

are discussed in Section V.

II. Mathematical Background

Modular exponentiation, which is used in public-key cryptography schemes such as RSA [8], requires multiplication and squaring within an integer field. These integers are potentially of many thousands of bits in length. ECC may be performed using different mathematical bases and the processor presented in this paper supports integer and polynomial basis. In contrast to exponentiation the field lengths used in ECC are typically of a few hundred rather than thousands of bits in length.

A. Exponentiation

Exponentiation is typically performed using a *square-and-multiply* algorithm, although other methods may be used [5]. This algorithm traverses the bits of the exponent and performs squaring and multiplication as appropriate in order to obtain the exponential. Modular exponentiation forms the mathematical basis of cryptographic schemes that utilise the integer factorisation problem [9] (e.g. RSA) or the discrete logarithm problem [10] (e.g. Diffie-Hellman key exchange [11]).

B. ECC

ECC is increasingly being found in security standards and practical implementations due to its advantages over exponentiation based methods. These advantages include lower memory requirements and key lengths for any given security strength and lower complexity arithmetic offered by polynomial and optimal normal basis over binary extension fields. Analogous to the *square-and-multiply* algorithm used in exponentiation, a scalar point multiplication of a point on an elliptic curve is performed using the *double-and-add* algorithm and the chord-tangent law [12].

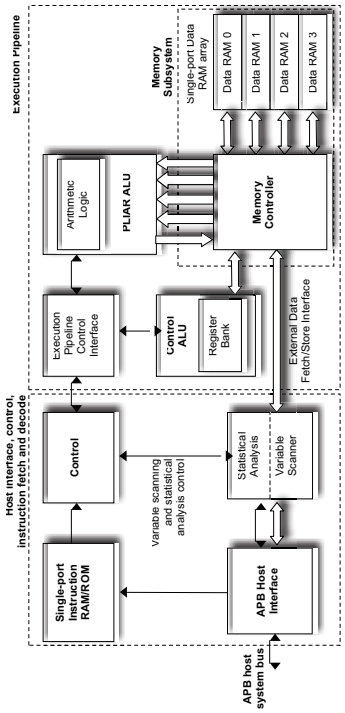
III. Asymmetric Processor Architecture

The public-key processor (PKP) has been designed as a high performance hardware solution to public-key cryptography that complements a general purpose host microprocessor. The transfer of complex and data intensive mathematical operations to the PKP allows such a host microprocessor to perform either additional functionality or the use of a more resource and/or power efficient host microprocessor. A block diagram of the general architecture is shown in Figure 1. The PKP has been developed as a Verilog RTL synthesizable core and may be scaled to allow the resources and performance to be balanced as appropriate.

A. Design Overview

The processor has been provided with an APB (Advanced Peripheral Bus) slave interface to allow its status, commands and data to be communicated with a host microprocessor using a standard system bus. A statistical analysis block attached to the APB interface allows bit composition measures to be gathered for any data written to the PKP local RAM. Such measures may be used to determine the applicability of a particular technique to improve performance. For example various integer recoding techniques are dependent on the particular bit distribution of the integer.

Fig. 1. Public-Key Processor Block Diagram



All instructions are stored in a 28-bit wide single-port RAM of sufficient size for the application. A range of registers are provided at the top-level for control purposes. A number of these registers are firmware defined and may be programmed as necessary. The instruction decode logic is used to fetch instructions from RAM at the requested address and determine how to execute that instruction. The simple high-level instructions that control function calls and configuration of specific functional blocks are 14 bits in length, while the more complex arithmetic and data manipulation instructions are 28 bits in length. The shorter high-level instructions are therefore packed together into a 28-bit space in order to reduce the required storage space. This allows a smaller instruction RAM resources to be utilized in an SoC design.

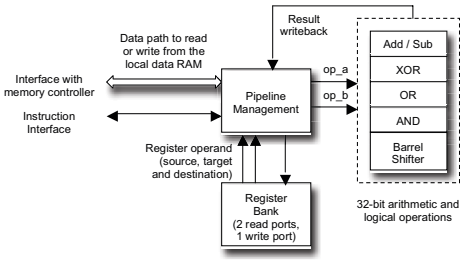
The execution pipeline of the PKP is composed of three distinct hardware blocks. These are the Control ALU, Polynomial and Large Integer Arithmetic (PLIAR) ALU and the local memory subsystem. Once decoded, VLIW commands are passed to either the Control or PLIAR ALU's. The Control ALU is capable of reading and writing to the local data RAM through the memory subsystem. It is also provided with access to all programmable control and status registers within the PKP as well as 8 data registers of its own. The PLIAR ALU uses the memory subsystem to access all multiple-precision operands in order to perform arithmetic operations. The use

of RAM rather than fixed length registers enables the PKP to operate over fields of arbitrary length.

The processor utilizes three groups of instructions which are carried out by three different execution paths. The high-level instructions are used to setup integer scanning hardware, generate software-driven interrupts and perform branch operations.

The Control ALU provides a range of instructions that may be used to perform bit-oriented data manipulation. The large numbers that are typically used on this processor may require fine granularity manipulation such as that required for integer recoding. The Control ALU also provides the functionality required to setup address pointers which are used to indicate where large numbers are located in the local memory subsystem. The control ALU is shown in Figure 2.

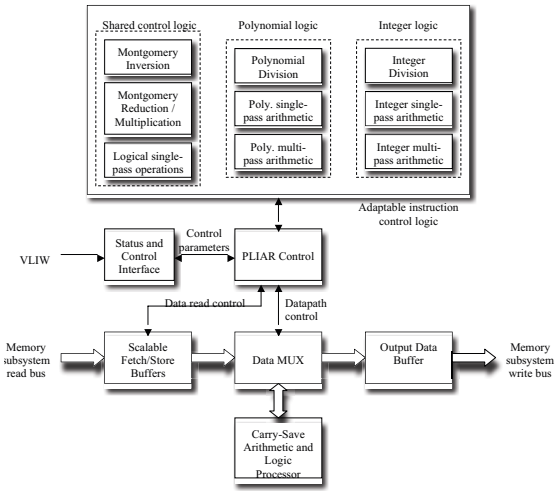
Fig. 2. Control ALU block diagram



The PLIAR ALU performs all integer and polynomial arithmetic. Operations such as Montgomery multiplication, Montgomery inversion, addition or division are performed by a single instruction. The number of cycles that each of these instructions consume is dependent on the arbitrary length of the numbers involved and scalable options within the processor such as the pipeline width and the memory subsystem configuration. As an example, the 32-bit *MEDIUM* configuration of the processor (described later) results in a Montgomery multiplication time of approximately $(m/16)^2$ cycles for both integers and polynomials, where m is the length of the modulus. It should be noted that performance is dependent on the memory subsystem and firmware that enables optimal burst access from synchronous RAM.

A block diagram of the PLIAR ALU is shown in Figure 3. All data processing is handled by an arithmetic processing block that contains all multiplicative and additive arithmetic and logical operations. Data is directed into this block from the client memory interface. This control logic is composed of a number of finite state machines that are under the control of the decoded VLIW command. Arbitrary field lengths may be supported by the control logic by defining the length of the field with control registers which the host microprocessor may program. The complex arithmetic such as Montgomery multiplication/reduction may use these control registers to define the length of various operands. The firmware may utilise these registers with the PLIAR instructions to provide a flexible and adaptable means of supporting arbitrary field lengths using the same firmware library.

Fig. 3. PLIAR ALU block diagram



B. Scalability

The architecture is scalable in terms of both the instruction set and the hardware within the execution pipeline that supports these instructions. For example, all of the integer instructions may be omitted if binary field ECC is foreseen as the only application. Alternatively, if an application such as RSA will be used exclusively all polynomial basis instructions may be omitted.

Hardware within the processor may be scaled or omitted entirely. Adjusting the instruction set will remove the associated control and arithmetic logic. It is possible to vary the execution pipeline width between 8, 16, 32, 64 and 128 bits. This will adjust the width of arithmetic hardware such as multipliers and adder trees as appropriate. The multipliers (both integer and polynomial) may be varied between full parallel and Karatsuba [13] multi-cycle implementations. Montgomery multiplication may be supported by optimised hardware using two multipliers in parallel as shown in Figure 4. To reduce hardware resources this functionality may be supported by Montgomery reduction proceeding multiple-precision multiplication which requires only a single multiplier as shown in Figure 5.

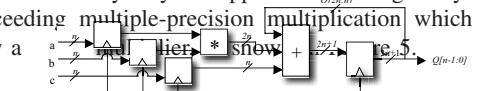
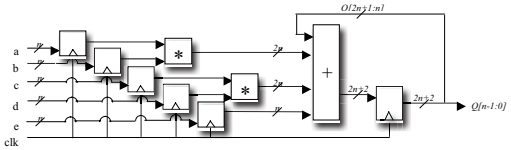
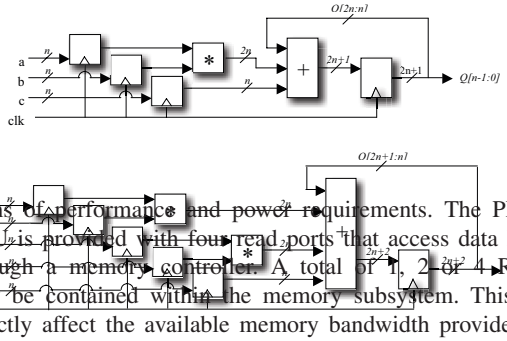


Fig Optimised Montgomery multiplier circuit block diagram



The local memory subsystem may also be varied in terms of its width (which varies with the execution pipeline width), the available memory depth and the number of parallel single-port RAMs. The latter option offers the greatest variation in

Fig. 5. Montgomery reduction circuit block diagram



terms of performance and power requirements. The PLIAR ALU is provided with four read ports that access data RAM through a memory controller. A total of 1, 2 or 4 RAMs may be contained within the memory subsystem. This will directly affect the available memory bandwidth provided for the PLIAR ALU.

The range of options provided to scale the processor architecture must be carefully balanced. For example, a resource efficient and relatively low performance PLIAR ALU should not be paired with a high performance memory subsystem.

IV. Performance Evaluation

In order to evaluate the performance of the processor a comparison is provided between the different schemes it supports. In addition, a comparison is provided with previous work reported in the literature. In all cases a fixed 32-bit pipeline width implementation is used. This comprises the full instruction set, optimised Montgomery multiplication and an optimal configuration of 4 parallel data RAMs within the memory subsystem.

A. Modular Exponentiation

The results shown in Table I illustrate the variation in performance offered by different *square-and-multiply* algorithms used in conjunction with various integer recoding schemes. It can be seen that *4-NAF* integer recoding offers the fastest computation time when used with *left-to-right* exponentiation, followed by the two *m-ary* methods. The *NAF* methods offer superior performance to binary exponentiation methods, but the large initialization phase of *NAF* and *w-NAF* is time-consuming. Therefore the impact of pre-computation must be carefully considered when determining the best method in a given situation. The extra memory requirements of the integer recoding methods must also be considered in a constrained system.

TABLE I

1024-bit field modular exponentiation performance using various square-and-multiply and integer recoding methods

Integer Recoding Scheme	Computation (clock cycles)			Integer recoding gain (excluding preprocessing)
	Pre	Exponentiation	Post	
L-R Binary	17898	3870400	2592	-
R-L Binary	17889	3896082	2591	-
L-R M-Ary	38422	3277709	2592	15.3 %
R-L M-Ary	19456	3277813	22012	14.8 %
L-R NAF	834401	3452544	2592	10.8 %
L-R 4-NAF	866299	3265005	2592	15.6 %

B. Prime Field ECC

Three different coordinate systems have been utilised in determining the performance of prime field ECC - affine, standard projective and Jacobian projective. The performance of these coordinate systems is shown in Table II. It is observed that standard projective coordinates offers a marginally better performance than the Jacobian projective system, while it may be no surprise that affine coordinates are highly inefficient due to the abundant use of field inversion. As such performance figures were obtained only for 192-bit GF(p) NIST curves. It should be noted that the affine coordinate system requires only 60% of the instruction RAM consumed by the projective methods due to the relative simplicity of its point doubling and point addition algorithms and the PKPs high-level arithmetic instructions which reduce the complexity of firmware.

TABLE II

Prime field ECC with various coordinate systems

NIST Curve	Coordinate system	Computation time (clock cycles)
P192	Standard Projective	820770
	Jacobian Projective	823972
	Affine	19776563
P224	Standard Projective	1147098
	Jacobian Projective	1157667
P256	Standard Projective	1799233
	Jacobian Projective	1817877
P384	Standard Projective	4082364
	Jacobian Projective	4127213

The performance of various integer recoding schemes when used with standard projective coordinates are shown in Table III. It can be seen that the same raw performance improvements are present as that observed in modular exponentiation. *NAF* and *w-NAF* methods require more pre-computation than the binary or *m-ary* methods. This pre-computation is in the form of an expensive field inversion required for every pre-computed point on the curve. When including the cost of pre-computation the *m-ary* method offers the fastest computation time, otherwise *w-NAF* is marginally the fastest method when pre-computation can be ignored. Knowledge of the application may aid in determining the most applicable method as some elements of pre-computation may be considered as a one-time setup. For example, Diffie-Hellman may re-use the same base point and field thereby allowing the pre-computation of a number of points on the field for various integer recoding methods.

C. Binary Field ECC

Four different coordinate systems were used to evaluate the performance of binary field ECC - affine, standard projective, Jacobian projective and López-Dahab projective. However, the performance using affine coordinates is vastly inferior to that of the projective systems. Table IV provides the average performance using these projective coordinate systems and a binary double-and-add algorithm. It is shown that Jacobian coordinates offer the fastest computation time. López-Dahab

TABLE III
PRIME FIELD ECC WITH VARIOUS INTEGER RECODING SCHEMES

NIST Curve	Integer Recoding Scheme	Computation (clock cycles)			Integer recoding gain (excluding preprocessing)
		Pre	Double-and-Add	Post	
P192	Binary	2216	788886	29668	-
	M-Ary	15050	703914	29405	9.0
	NAF	108870	698732	29665	12.9
	4-NAF	123170	690573	29525	12.6
P224	Binary	2722	1105351	39024	-
	M-Ary	17998	974360	39734	9.0
	NAF	134796	974549	39150	13.4
	4-NAF	151782	968807	39301	12.9

followed by standard projective coordinates are marginally slower. It follows that the increased computation and storage requirements of Jacobian coordinates provides for an optimal computation time. The use of affine coordinates unsurprisingly offers the longest computation time and lowest use of both instruction and data RAM due to its low complexity.

The performance gains offered by integer recoding in binary field ECC are very similar to that offered by integer recoding when used with ECC over prime fields. Table V shows the performance gains offered by integer recoding and Jacobian coordinates. The windowed *NAF* method offers a nominally better performance than the other integer recoding methods. The two *NAF* methods require relatively expensive field inversions to be performed as part of the pre-computation step. When the base point and curve are repeatedly used this field inversion may be considered as a one-time setup operation and ignored.

TABLE IV
BINARY FIELD ECC WITH VARIOUS COORDINATE SYSTEMS

NIST Curve	Coordinate system	Computation time (clock cycles)
B111	Standard Projective	261201
	Jacobian Projective	253965
	López-Dahab Projective	258253
B163	Standard Projective	608658
	Jacobian Projective	593822
	López-Dahab Projective	598345
B233	Standard Projective	1286818
	Jacobian Projective	1258493
	López-Dahab Projective	1260072

D. Comparison to related work

The relative performance of the processor in comparison to previous work presented in the literature and commercial solutions is shown in Table VI. The range in performance of the processor's scalable execution pipeline is demonstrated using five common measurements - 192-bit and 1024-bit integer field multiplication, 163-bit polynomial field multiplication, 192-bit integer field addition and 163-bit polynomial field addition. Three scaled implementations of the processor were chosen to compare the resource and performance metrics:

TABLE V
BINARY FIELD ECC WITH VARIOUS INTEGER RECODING SCHEMES

NIST Curve	Integer Recoding Scheme	Computation (clock cycles)			Integer recoding gain (excluding pre-processing)
		Pre	Double-and-Add	Post	
B111	Binary	24406	214262	15296	-
	M-Ary	32564	183899	15014	7.2
	NAF	77698	179890	15212	19.1
	4-NAF	86728	174942	14903	22.5
B163	Binary	56581	506926	30315	-
	M-Ary	69201	426795	30284	5.2
	NAF	159197	423800	30211	19.6
	4-NAF	173170	416956	30657	21.6
B233	Binary	119372	1082903	56218	-
	M-Ary	137126	887423	55582	5.4
	NAF	306861	895839	56560	20.9
	4-NAF	326446	885300	55467	22.3

Compact	8-bit pipeline
	1 x 2048x8 single-port data RAM
	Supports up to 2040-bit fields
Medium	32-bit pipeline
	2 x 512x32 single-port data RAMs
	Supports up to 2016-bit fields
Ultra	128-bit pipeline
	4 x 256x128 single-port data RAMs
	Supports up to 3968-bit fields

These all utilise both integer and polynomial instructions, optimised Montgomery multiplication support and a 512x28 word instruction RAM. Note that the *Medium* implementation has a reduced memory bandwidth compared to the 32-bit implementation used in the previous performance analysis.

Table VI illustrates the variation in performance and resource usage of the public-key processor presented in this paper. The low power and resource efficient *compact* implementation of the processor is shown to offer high performance and efficient resource usage. While the 8-bit parallel Montgomery multiplier circuit offers high performance it is not a low power design in comparison to [15]. However the resource usage of the *Compact* design is efficient by comparison.

The Domain Specific Reconfigurable Cryptographic Processor (DSRCP) [17] is capable of the same operations. However, the PKP offers a scalable architecture for SoC integration, is not limited to a maximum field size of 1024 bits and offers software definable integer recoding rather than a hardcoded scheme. Furthermore, the PKP is not limited to the affine ECC coordinate system and offers a complete solution to perform ECC point multiplications over prime fields that does not require external resources. No hardware resources for the DSRCP are stated in order to make a fair comparison of performance.

Also of relevance is [16] which describes a dual-field ECC processor. This shares many design features of the PKP in a compact architecture that outperforms our implementation. For example, our 32-bit 90.9K gate (logic) PKP compared to their 32-bit 43.52 K gate (logic) processor performs ECC 160-bit point multiplication in 3.94 ms *GF(p)* and 3.72 ms *GF(2^n)*

TABLE VI
RESOURCE AND PERFORMANCE COMPARISON

Version	Resource metrics	Operation	Computation time
This work - *COMPACT*	200MHz TSMC 130 nm (worst case) 42.7k logic gates 36.5k RAM gates	INTEGER: 1024-bit mult. 192-bit mult. 192-bit addition POLYNOMIAL: 163-bit mult. 163-bit addition	 347.5 μs 13.9 μs 770 ns 10.7 μs 340 ns
This work - *MEDIUM*	166 MHz TSMC 130 nm (worst case) 90.9k logic gates 34.9k RAM gates	INTEGER: 1024-bit mult. 192-bit mult. 192-bit addition POLYNOMIAL: 163-bit mult. 163-bit addition	 20.31 μs 1.04 μs 126 ns 840 ns 90 ns
This work - *ULTRA*	133 MHz TSMC 130 nm (worst case) 339 k logic gates 51 k RAM gates	INTEGER: 1024-bit mult. 192-bit mult. 192-bit addition POLYNOMIAL: 163-bit mult. 163-bit addition	 2.20 μs 98 ns 83 ns 98 ns 60 ns
[14]	0.5 μm CMOS 27k logic gates 80 MHz, Mult. circuit only	1024-bit integer multiplication	43 μs
[15]	0.18 μm CMOS 59k gates, 2kB RAM, 50 MHz, Mult. circuit only, low power	1024-bit integer multiplication	180 ms
[16]	0.13 μm CMOS 32-bit version 178.6 MHz	256-bit int mult 160-bit poly mult	190 cycles 72 cycles
[17]	FPGA, 50 MHz resources not stated	1024-bit int mult 192-bit poly mult	2048 cycles 31.25 μs

compared to 1.71 ms *GF(p)* and 0.34 ms *GF(2^n)* respectively. Our design is a firmware driven processor and not a sequencer driven hardware implementation that uses fixed support for algorithms. This results in the increased gate count and lower performance of the PKP. However, our design offers much greater flexibility as seen by its greater support of different ECC and modular exponentiation approaches.

V. CONCLUSION

This paper describes a processor architecture for public-key cryptography applications. A relatively high performance 32-bit implementation of the processor is used to illustrate the performance advantages when using different mathematical optimisation techniques. These techniques are provided as firmware that may be downloaded to the instruction RAM. Such reprogrammable functionality is a unique feature of the described architecture. Such flexibility enables the host system to randomly select from a range of integer recoding, coordinate systems and *double-and-add* algorithms for every ECC point multiplication in order to offer resistance to timing attacks.

The processor is capable of supporting integer and polynomial arithmetic. Optimised instructions are provided to enable highly efficient multiplication, addition and inversion within fields. Such arithmetic is necessary for supporting many public-key cryptography methods which include exponentiation and ECC point multiplication over both prime and binary fields. The supported field and operand lengths are only restricted by the data RAM provided in the design, contrary to many other published designs that have fixed field lengths. The design is also not restricted to any particular parameters of the exponentiation or ECC scheme being performed.

An ASIC in the field utilising this application specific processor could be adapted using new firmware to support different applications. The instruction set of the processor may also allow the processor to adapt to new public-key cryptography techniques as they evolve from experimental methods. Future work will evaluate the processors ability to support hyperelliptic- and torus-based cryptography and support for hybrid *GF(p)* and *GF(2^n)* arithmetic hardware.

ACKNOWLEDGMENT

This work was sponsored by Conexant Systems Inc.

REFERENCES

[1] N. Koblitz, *"Elliptic curve cryptosystems,"* Mathematics of Computation, vol. 48, pp. 203209, 1987.
[2] V. Miller, *"Use of elliptic curves in cryptography,"* CRYPTO 85, 1985.
[3] P. L. Montgomery. *Modular multiplication without trial division*. Mathematics of Computation, vol. 44, no. 170, pp. 519-521, 1985.
[4] P. Barrett, *Implementing the Rivest Shamir and Adleman public key encryption algorithm on a standard digital signal processor,* in *Advances In Cryptology CRYPTO 86 (LNCS 263)* (A. M. Odlyzko, ed.), pp. 311323, 1987.
[5] A. Menezes, P. van Oorschot, and S. Vanstone, *Handbook of Applied Cryptography. CRC Press*, 1996. http://www.cacr.math.uwaterloo.ca/hac/.
[6] N. Smyth, M. McLoone and J. McCanny, *"Reconfigurable Processor for Public-Key Cryptography",* SIPS 2005, Proceedings, IEEE International Conference on 2-4 November 2005, pp. 110-115.
[7] B. S. Kaliski Jr., *T he Montgomery inverse and its applications,* IEEE Transactions on Computers, 44(8), pp. 10641065, Aug 1995.
[8] PKCS #1: RSA Cryptography Standard, www.rsasecurity.com/rsalabs/pkcs/pkcs-1/, March 2006.
[9] D. Knuth. *The Art of Computer Programming*, Volume 2: Seminumerical Algorithms, Third Edition. Addison-Wesley, 1997. ISBN 0-201-89684-2. Section 4.5.4: Factoring into Primes, pp. 379417.
[10] *Discrete logarithm*, Wikipedia, the free encyclopedia, http://en.wikipedia.org/wiki/Discrete_logarithm, March 2006.
[11] W. Diffie and M. E. Hellman, *New Directions in Cryptography*, IEEE Transactions on Information Theory, vol. IT-22, Nov. 1976, pp. 644-654.
[12] M. Rosing, *Implementing Elliptic Curve Cryptography*, K. Antonsen, Ed. Manning Publications Co., 1999.
[13] A. Karatsuba and Y. Ofman, *"Multiplication of Many-Digital Numbers by Automatic Computers,"* Doklady Akad. Nauk SSSR 145, 293-294, 1962. Translation in Physics-Doklady 7, 595-596, 1963.
[14] A. F. Tenca and C. K. Koç, *A Scalable Architecture for Modular Multiplication Based on Montgomerys Algorithm,* IEEE Transactions on Computers, Vol. 52, No. 9, September 2003.
[15] Hee-Kwan Son and Sang-Geun Oh , *Design and implementation of scalable low-power montgomery multiplier,* Computer Design: VLSI in Computers and Processors, 2004. ICCD 2004. Proceedings. IEEE International Conference on 11-13 Oct. 2004 pp. 524 531.
[16] A. Satoh and K. Takano, "A Scalable Dual-Field Elliptic Curve Cryptographic Processor," IEEE Trans. Computers, vol. 52, pp. 449-460, 2003.
[17] J. Goodman and A. P. Chandrakasan, "An energy efficient reconfigurable public-key cryptography processor architecture," In Proceedings of 2nd International Workshop on Cryptographic Hardware and Embedded Systems (CHES), number 1965 in Lecture Notes in Computer Science, pages 174-191, Worcester, Massachusetts, USA, August 17-18 2000. Springer-Verlag.