MicroECC: A Lightweight Reconfigurable Elliptic Curve Crypto-Processor

Michal Varchola
ELIT SYSTEMS, s. r. o.,
Technical University of Košice
Košice, Slovak Republic
michal@varchola.com

Tim Güneysu
Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
tim.gueneysu@rub.de

Oliver Mischke
Horst Görtz Institute for IT-Security
Ruhr-University Bochum, Germany
oliver.mischke@rub.de

***Abstract*—In this paper we present compact FPGA-based architectures for standardized elliptic curve cryptography over prime fields. Our approach differs from the many previous works due to the following design principles: First, we minimized storage by efficiently using block memories instead of registers, and second, we focused on elliptic curves based on standardized NIST primes. Furthermore, the presented MicroECC processors are optimized for two goals: a first architecture utilizes a 16-bit data path and a single 16-bit hardware multiplier and is optimized for minimal FPGA resource consumption. The second processor design employs a 32-bit data path and several hardware multipliers for improved throughput. Both implementations are not fixed to a single curve and support point multiplications for (but not limited to) both NIST curves P-256 and P-224. Tested on Xilinx and Microsemi FPGAs, our ECC-P256 processors provide a significantly better performance-per-slice ratio (i.e., a factor of 7.1 and 6.3 for the 16-bit and 32-bit architecture, respectively) compared to a comparable implementation, recently presented on ASAP 2010.**

***Keywords*-Elliptic Curve Cryptography, NIST Primes**

I. Introduction

Neal Koblitz and Victor Miller proposed in 1985 [1], [2] the use of Elliptic Curve Cryptography providing an equivalent level of security compared to classical asymmetric cryptosystems although using smaller keys (i.e., parameters with a typical size between 160–256 bit [3]). Therefore, ECC has become the preferred asymmetric cryptosystem for many products, demonstrated in several standards by IEEE, ANSI and SECG [4], [5], [6], [7]. Concurrently, Field Programmable Gate Arrays (FPGA) have evolved to a powerful alternative with respect to classical VLSI and ASIC circuits during the last years. FPGAs provide the invaluable advantage of dynamic and flexible circuit reconfigurability allowing for high-performance hardware implementation with little development efforts and costs. For this reason, FPGAs have emerged to a suitable hardware platform for the complex arithmetic typically used in asymmetric cryptosystems.

However, hardware implementations for ECC-based cryptography still requires a vast amount of resources due to their computational complexity. The work by Orlando and Paar [8], for example, requires more than 11,416 LUTs, 5,735 Flip-Flops and 35 BRAMs for ECC based on NIST's prime P-192 of a Xilinx XCV1000. Such a large design even takes a significant portion of large (and thus expensive) FPGA devices. Therefore, our primary design goal in this work is to develop a small, flexible and resource-efficient architecture for standardized elliptic curve cryptography on prime fields. Such a cryptographic architecture can be built into a main application that occupies already large parts of the FPGA device. Moreover, our designs were developed to comply to standards and easily adopt countermeasures against side-channel attacks.

Contribution: In this work, we propose an ECC architecture that is not only smaller than latest most compact ECC implementations over prime fields (i.e., [9]), but at the same time up to 7.1 times more efficient. However, it needs to be remarked that we restrict our design to use primes in a special standardized form, while [9] supports general elliptic curves by using the Montgomery multiplication algorithm [10]. But our ECC engine is not limited to just a single curve. In fact, we can switch security parameters during runtime, such as the prime of the underlying finite field. Part of this work are designs to run operations on elliptic curves using standardized NIST primes P-256 and P-224 (which will be denoted as ECC-P224 and ECC-P256 respectively in the following sections of this work). Additional curves can be supported by reloading of the embedded memory without requiring additional changes on the underlying hardware. We primarily focused on the curves ECC-P256 and ECC-P224 due to their practical relevance for industry applications such as the upcoming IEEE 1609 standard and provide results both for Xilinx and Microsemi (previously Actel) FPGAs.

Outline: This work is structured as follows. In Section II we briefly discuss previous work with relevance to this contribution. Section III provides a short introduction to elliptic curves based on the NIST FIPS 186–3 standard. Sections IV and V describe our architecture and corresponding implementation. Our results are presented in Section VI before we conclude in Section VII.

II. Previous Work

For ECC, the most common choice for the underlying finite field is either binary fields with prime extension, denoted by GF(2^m), or prime fields, denoted by GF(p).

978-0-7695-4551-6/11 $26.00 © 2011 IEEE
DOI 10.1109/ReConFig.2011.61

Binary fields provide an intrinsic advantage for hardware implementation because the field addition does not require costly carry computations. However, with respect to actual cryptographic protocols such as the Elliptic Curve Digital Signature Algorithm (ECDSA) a prime field is still often preferable since arithmetic operations can be reused for other GF(p) operations. Benefits of the micro-code approach, firstly introduced in [11], are also used in our design, however we will exclusively focus on standardized NIST recommend prime curves. With NIST's ECC-P256 curve, we not only achieve a high level of security but also allow for a fair comparison to recently published cores (cf. Section VI).

The previously most compact ECC core for prime fields was presented by Vliegen *et al*. [9]. By using a small number of hard macros like BRAMs and dedicated multipliers of Xilinx FPGAs (which are nowadays common in almost all current FPGA families), and using a micro-code approach, they were able to drastically decrease the amount of required logic, and build a very small but still fast enough circuit.

One of the fastest implementations for elliptic curves over the 256-bit NIST prime was presented at CHES 2008 by Güneysu and Paar [12]. To achieve its high performance it applies a large number of 32 DSPs (used as Multiply-and-Accumulate unit) and BRAMs on Virtex-4 FPGAs.

In this work, our goal was to combine both goals mentioned above, i.e., create a more compact implementation than [9], but still achieve a better area-time product than the high-performance implementations presented in [12].

III. Background

In this section, we introduce the basic mathematical background related to implementations of arithmetic instruction sets and ECC algorithms. As mentioned above we designed the ECC architecture for use with prime fields GF(p), where p denotes a standardized NIST prime. We will now shortly review the basics required to build ECC cryptosystems.

Let S be a non-infinite point on an elliptic curve with Weierstrass equation $\epsilon: \; y^2 = x^3 + ax + b \pmod p$, defined over Galois Field GF(p) where $p > 3$ is a prime. Additionally, parameters $a, b \in$ GF(p) need to fulfill $4a^3 + 27b^2 \neq 0$ to prevent the partial derivatives of the curve from vanishing. The affine representation of S is a pair $S = (x, y)$ of elements of GF(p) which satisfy the defining equation ϵ. The ECC arithmetic defines the addition of two points $R = S + T$ based on a group law using the *tangent-and-chord* rule. Since the formulas for point addition on Weierstrass curves are not complete, we need to use two different algorithms: point doubling (if $S = T$) and point addition (if $S \neq \pm T$).

The group law in affine representation requires a costly modular division (inversion) for both point doubling and point addition. However, we can omit the modular inversion operation when using a projective representation of the curve. The projective coordinates of the point on the curve is given by the triple $S = (X, Y, Z)$ with coordinates X, Y, Z in GF(p). To convert a point S between affine and projective coordinates we can apply the relationship $x = X/Z^\lambda$ and $y = Y/Z^\sigma$ with $\lambda = \sigma = 1$ for standard projective and $\lambda = 2$ and $\sigma = 3$ for Jacobian projective coordinates, respectively. For projective representations, point addition and doubling can be computed using a series of modular additions, subtractions and multiplications.

The security of all ECC cryptosystems relies on the difficulty of Elliptic Curve Discrete Logarithm Problem (ECDLP). The ECDLP considers the efficient computation of an unknown parameter k from two given points R, S with $R = k \cdot S$ and $k \in$ GF(p) and $R, S \in \epsilon$ as a hard problem. In this context, the operation $k \cdot S$ is known as point multiplication and practically means k-repeated point additions of a given base point S. The ECDLP is the fundamental cryptographic problem used in cryptographic protocols, schemes and algorithms such as Elliptic Curve Diffie-Hellman key exchange, ECIES encryption scheme [13], and the Elliptic Curve Digital Signature Algorithm (ECDSA) [5].

In this work, we need to implement a modular arithmetic unit for elliptic point operations based on the prime field GF(p). The basic arithmetic computations, such as addition and multiplication, always include a subsequent step to reduce the result to the domain of the underlying field. Since the reduction is costly for common prime fields since it involves multi-precision division, special primes have been proposed by Solinas [14] providing efficient reduction algorithms based on a sequence of multi-precision addition and subtractions only. These primes denoted by P-l with $l = \{192, 224, 256, 384, 521\}$ specifying a corresponding bit size, were standardized by NIST in [15].

Algorithm 1 presents the modular reduction for P-256 as used in this work, requiring two doublings, four 256-bit subtractions and four 256-bit additions. Based on the computation $Z = z_1 + 2z_2 + 2z_3 + z_4 + z_5 - z_6 - z_7 - z_8 - z_9$, the range of the possible result is $-4p < Z < 5p$ which needs to be corrected in a final step.

A second major aspect of this work is the resistance of our solution against Simple Power Analysis (SPA) and Differential Power Analysis (DPA) as introduced by Kocher in [16]. These techniques enable to reveal a secret information contained in the implementation of a cryptosystem by monitoring side-channel information such as power traces. The former investigates the shape or duration of single power trace of a single execution of the cryptographic algorithm. Particularly, the school-book left-to-right algorithm for a point multiplication $k \cdot S$ is highly vulnerable due to different runtimes, depending on individual bits of the secret parameter k (i.e., if bit $k_i = 1$, this involves a longer computation than $k_i = 0$). In order to prevent such simple SPA attack, point multiplications should use uniform and timing-independent point multiplication algorithms, such as the Montgomery ladder [10].

The DPA attack is a more sophisticated method to cor-

Algorithm 1 NIST Reduction with P-256 $= 2^{256} - 2^{224} + 2^{192} + 2^{96} - 1$

```
Require: Double-sized integer $c = (c_{15}, \ldots, c_2, c_1, c_0)$ in
    base $2^{32}$ and $0 \le c <$ P-256$^2$
Ensure: Single-sized integer $c \bmod$ P-256.
1: Concatenate $c_i$ to following 256-bit integers $z_j$:
       $z_1 = (c_7, c_6, c_5, c_4, c_3, c_2, c_1, c_0)$,
       $z_2 = (c_{15}, c_{14}, c_{13}, c_{12}, c_{11}, 0, 0, 0)$,
       $z_3 = (0, c_{15}, c_{14}, c_{13}, c_{12}, 0, 0, 0)$,
       $z_4 = (c_{15}, c_{14}, 0, 0, 0, c_{10}, c_9, c_8)$,
       $z_5 = (c_8, c_{13}, c_{15}, c_{14}, c_{13}, c_{11}, c_{10}, c_9)$,
       $z_6 = (c_{10}, c_8, 0, 0, 0, c_{13}, c_{12}, c_{11})$,
       $z_7 = (c_{11}, c_9, 0, 0, c_{15}, c_{14}, c_{13}, c_{12})$,
       $z_8 = (c_{12}, 0, c_{10}, c_9, c_8, c_{15}, c_{14}, c_{13})$,
       $z_9 = (c_{13}, 0, c_{11}, c_{10}, c_9, 0, c_{15}, c_{14})$
2: Compute $c = (z_1 + 2z_2 + 2z_3 + z_4 + z_5 - z_6 - z_7 -$
    $z_8 - z_9 \bmod$ P-256)
```

relate a hypothetical power estimation for a key guess with a large number of power traces. DPA statistically evaluates many executions of the cryptographic algorithm with different inputs. There are several techniques to protect the ECC computations against DPA, such as exponent randomizations and Z-coordinate randomization if projective coordinates are used. A more detailed survey on SPA and DPA countermeasures for ECC are given in [17], which we can apply within the micro code of our ECC architecture as shown in the next section.

IV. Designing a Compact ECC Architecture

As could be seen above, we need to efficiently implement group operations on elliptic curves, such as point doubling and point addition, which are used in point multiplication algorithms. Modular inversion is computed using Fermat's little theorem. According to point addition formulas, we need to implement computations on prime fields, namely modular additions, subtractions and multiplications.

A first approach could be to implement the entire ECC group operations in hardware. Such an approach usually results in a quite complex design approach involving lots of hardware resources. A second idea is to use a standard off-the-shelf microprocessor with a given, predefined instruction set to implement the arithmetic operations. Obviously, a suitable trade-off between both approaches could be to implement specialized underlying instructions in hardware and control their execution in proper order by an embedded program executed on a simple, special-instruction microprocessor. This general concept is shaping the MicroECC architecture as shown in Figure 1.

A. *Modular Arithmetic and Logic Unit*

The modular Arithmetic and Logic Unit (ALU) span the hardware part of MicroECC and supports basic 256-bit prime field arithmetic instructions. According to the formulas and requirements presented in Section III, the implemented instructions are:

- Modular addition, subtraction and multiplication
- Comparisons to $a > b$, $a = b$ (which are used, e.g., in conditional loops)
- I/O operations between data memory and an external interface
- Bit extraction from a 256-bit operand required by point multiplication

MicroECC does not use any 256-bit registers synthesized in logic for storing operands or results. All constants, input data, final and intermediate results are stored in data memories (DM A and DM B).

To reduce the system footprint, the data path of the modular ALU is 16-bit or 32-bit wide, which implies 16 or 8 words, respectively, to store a 256-bit operand. We will denote the data path width as W in the following. All field computations are executed word-wise using W-bit operand blocks. We use two separated dual port RAMs (DM A and DM B) in order to provide four separate ports, which enable to read two operand words and the prime number as well as to store a resulting word from a previous operation per clock cycle. However, the usage of two memories adds extra complexity to the MicroECC because we have to specify particular memory blocks for each operand and result. On the other hand, the performance can be doubled with respect to a conventional *load-and-store* architecture that uses a single data memory block.

The Data Memory Controller (DM CTRL) is a set of multiplexers to switch between both DMs to load and store each operand or result independently. The address and data buses are enabled to selected memory block depending on the value of the *DM A/B* signal for both operands (*opA* and *opB*), prime (p) and *result*. All word-wise computations are performed by the W-bit ALU (Figure 2). The ALU Controller (ALU CTRL) takes care of the operand processing of all word-wise operations in proper order. The operation of both the ALU and the ALU CTRL units are explained in Section V.

Each DM can store up to $2^A \times W$-bit words (e.g. $A = 10$ if $W = 16$ or $A = 9$ if $W = 32$). The address space is split into a lower address space of L bits (where $L = 4$ if $W = 16$ or $L = 3$ if $W = 32$) and a higher address space of H bits (e.g. $H = 6$). The former defines which word of an 256-bit operand is selected and the latter defines which operand is selected. Thus $A = H + L$.

B. *Software Engine*

We now discuss the software part of our ECC implementation. For this purpose we designed our own very simple

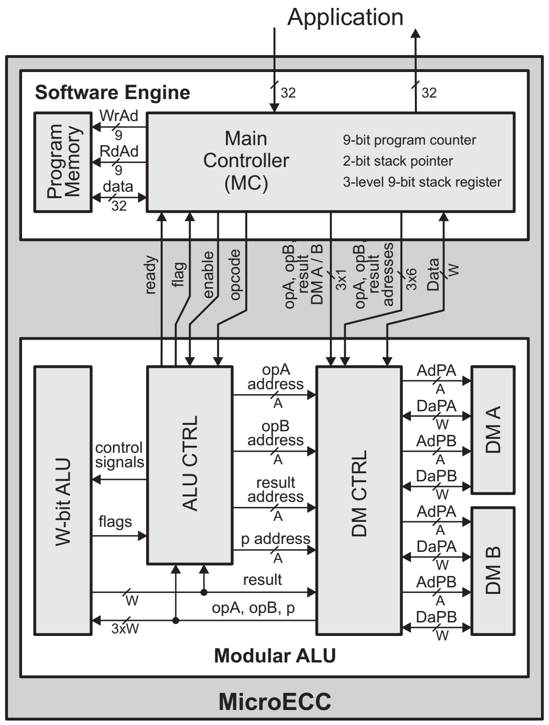

Figure 1. The MicroECC architecture consists of the Software Engine and the Modular Arithmetic and Logic Unit (Modular ALU)

microprocessor in FPGA logic. This processor is denoted as Software Engine in Figure 1 which combines a Main Controller (MC) and the Program Memory (PM). It performs the following tasks:

- Decode and execute commands coming from the application interface
- Transfer embedded programs into PM
- Execute embedded programs:
  - by controlling the program flow by instructions related to routine calls or returns, as well as conditional and unconditional jumps
  - by decoding the arithmetic and logic instructions which are executed in the Modular ALU

Commands received from the application interface issue a data transfer to PM or DM, or start the execution of an embedded program from a defined address. After an embedded program has been successfully executed, the MC returns to an idle state. The program flow is controlled by conditional or unconditional jumps, as well as routine calls and routine returns. We specified an instruction size of 32-bit since we need to include:

- 3×8-bit addresses of two operands and the result
- 3×1-bit selector between DM A or DM B for both operands and result
- 4-bit instruction code
- 1-bit selector that distinguishes whether the instruction is related to the program flow (executed on the MC) or field arithmetic (executed on the Modular ALU)

Thus a single BRAM with 32-bit data bus can store up to 512 instructions. An embedded program for point multiplication, which uses projective coordinates, consumes less than one quarter of a BRAM. The maximum number of 512 instruction implies the 9-bit program counter in the MC. Next, the default configuration of the MC includes a 2-bit stack pointer and a 3-level stack register. This allows to develop programs making use of up to three nested sub-functions.

We simplified the development of embedded programs for the MicroECC by designing our own assembler. Thus we can write an assembly code instead of machine code for the processor. The assembler is written in ANSI-C. The ANSI-C code also includes the MicroECC simulator that enables us to validate the proper functionality of newly designed MicroECC assembly code before any testing on real hardware.

C. Execution of Modular Arithmetic Instruction

Now we describe the execution of a single instruction on the modular ALU. The program counter points at the instruction code in the PM. The MC then checks that this instruction is related to the modular ALU. If so, the MC extracts up to 8-bit addresses of both the operands and the result inside the DM and their location in DM A or DM B, respectively. Then the MC issues an *enable* signal which triggers the Modular ALU to start the computation.

The addresses of both operands and result (higher H bits) are defined by the instruction opcode. The set of finite state machines (FSMs) contained in ALU CTRL generates word addresses in proper order and determines the particular W-bit word for the current operation. However, if storing or loading of intermediate results (e.g. the result of an addition before modular reduction) is required, the ALU CTRL can also generate the needed higher H bits of addresses. The group of FSMs in ALU CTRL also generates *control signals* for the W-bit ALU and can thus perform modular additions/subtractions and multiplications. When the modular ALU finished the computation, it issues a *ready* signal. The modular ALU can also provide the result of a comparison or the value of an extracted bit by a specific *flag* signal, which is useful for conditional jumps. Modular addition or subtraction and modular multiplication are frequently used in all ECC computations and therefore have an great affect on the performance of the core. We provide detailed implementation descriptions of each of them in the following section.

V. Implementation

In this section we will discuss the implementation details of MicroECC.

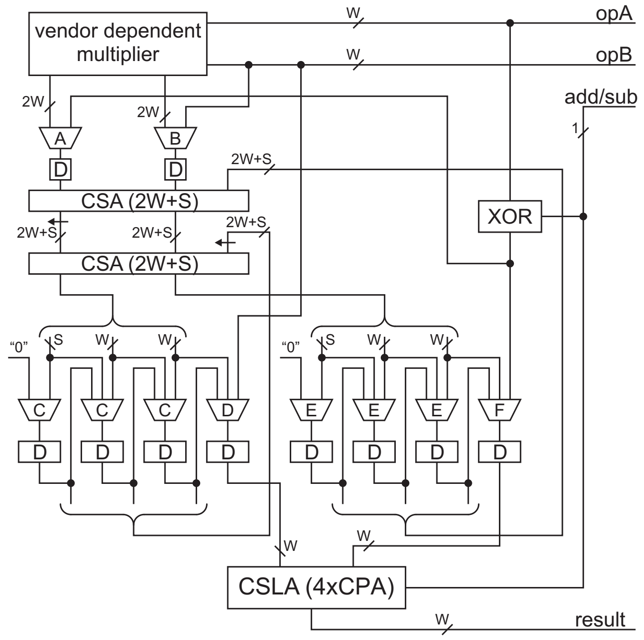

Figure 2. Schematic showing the W-bit Arithmetic-Logical Unit (ALU), where the accumulator is based on Carry Save Adders

A. Modular Addition and Subtraction

Modular addition consists mainly of two parts: multi-precision integer addition based on W-bit words and the subsequent modular reduction. Addition uses a W-bit Carry Select Adder (CSLA) composed of four $W/4$-bit Carry Propagate Adders (Figure 2). The special architecture of the adder allows a short critical path. The output of the W-bit CSLA (4xCPA) is routed back to the DM CTRL. The same signal is also fed into the comparator to test if a subsequent reduction is necessary or not (in case that the result is larger than the prime). If no reduction is required, the modular reduction subtracts zero from the result, otherwise the prime using the same addition hardware. Modular subtraction works similarly, but here we test whether the integer result becomes negative after subtraction. If the result still remains positive, no reduction is necessary and we add zero, otherwise the prime is added to the result.

B. Modular Multiplication

The modular multiplication algorithm (Algorithm 2) consists of three parts: *i)* integer multiplication, *ii)* the Fast Reduction (FR) algorithm, and *iii)* the final correction. The integer multiplication uses the operand-scanning method. The operand-scanning method utilizes a vendor-dependent hardware multiplier and accumulator. The multiplier has two outputs: a result vector and a carry vector as a result from Carry Save Addition (CSA). Both target FPGAs from Xilinx and MicroSemi provide different options to accelerate integer multiplication. The Xilinx Virtex-II Pro includes 18×18 hardware multipliers (of which one multiplier is used for $W = 16$ and four multipliers for $W = 32$), while the Microsemi SmartFusion does not offer hardware support. Therefore the $W \times W$ multiplier is synthesized in logic (tiles). The accumulator consists of two CSAs of $2W + S$ data-width and multiplexers C and E with registered outputs, where S is summation overhead when computing integer multiplication. The entire accumulation process is carried out using CSAs (in order to reduce the critical path) of which the result and carry vectors are finally added. For this, we use the same CSLA adder as mentioned above.

The FR described by Algorithm 1 adds and subtracts nine 256-bit integers z_y, where $y = 1..9$. Each integer z_y is concatenated from eight 32-bit words of the integer product. However, z_2 and z_3 is added twice and hence we will consider $y = 1..11$ in the remainder of this section. As our architecture performs W-bit operations, we assume z_y to be concatenated from N W-bit words of the integer product c_i, where $i = 0..2N - 1$. The sequence of consecutive c_i additions or subtractions is controlled by commands stored in DMs. We use the same accumulator as being used for integer multiplication, however, operands are routed directly to the accumulator input. Thus we can add/subtract and accumulate two c_i in a single clock cycle.

The last step is the final correction. As mentioned in Section III, the result of the FR should fall into the interval $-4p < r < 5p$ for the NIST P256 curve and so multiples of p have to be added or subtracted. All multiples of the prime are stored in the DM. Depending on the value remaining in the accumulator after the FR algorithm is finished (i.e., the remaining overflow after accumulation), one of the pre-computed multiples of the prime is selected and consecutively added to or subtracted from r. Note that the proposed implementation of the FR algorithm can handle all NIST primes by simply modifying the DM content.

VI. Results and Comparison

For our work we chose a Xilinx Virtex-II Pro XC2VP7-5FG456 and a Microsemi SmartFusion A2F500M3G-STD as target platform. A detailed comparison of our implementation results with [9] is shown in Table I. This allows for a fair comparison, since both ECC processors use a W-bit computational unit operating on up to 256-bit operands. Note that our design incorporates a communication interface so we have taken those results from [9] which also includes the communication interface. A comparison to other recent ECC FPGA implementations is given by Table II. At this point is worth to point out that our design and [9] are complementary rather than competitive, because former design is restricted to NIST primes whereas latter can use arbitrary prime.

Compared to the work by [9], our design requires less clock cycles and can be clocked at a higher frequency. Hence, the performance of a point multiplication is approximately three times better. Furthermore, we need approxi-

Table I
DETAILED COMPARISON OF THIS WORK VERSUS [9] IN TERMS OF AREA AND SPEED OF EXECUTION OF FOLLOWING OPERATIONS: ELLIPTIC CURVE POINT MULTIPLICATION (ECMP), MODULAR INVERSION (MI), POINT ADDITION AND POINT DOUBLING (PAPD), MODULAR MULTIPLICATION (MM), MODULAR ADDITION AND SUBTRACTION (MAS).

Design	**This Work**				**[9]**		**This Work**			
Device	**Xilinx Virtex-II Pro**						**Microsemi SmartFusion**			
Curve	**P224**	**P256**	**P224**	**P256**	**any**	**any**	**P224**	**P256**	**P224**	**P256**
Mode	**16-bit**		**32-bit**		**16-bit**	**32-bit**	**16-bit**		**32-bit**	
Max. Clk.[MHz]	210.0	210.0	210.0	210.0	108.2	68.17	109	109	109	109
Logic	773	773	1158	1158	1832	2085	3690	3690	7832	7832
RAM Blocks	3	3	3	3	9	9	12	12	12	12
HW MULTs	1	1	4	4	2	7	0	0	0	0
ECPM [cycles]	1722088	2103941	765072	949951	3227993	1074625	1722088	2103941	765072	949951
ECPM [ms]	8.20	10.02	3.64	4.52	29.83	15.75	15.80	19.30	7.01	8.91
MI [cycles]	189998	187694	85773	89078	247253	79141	189998	187694	85773	89078
PAPD [cycles]	6584	7729	2841	3171	11621	3853	6584	7729	2841	3171
MM [cycles]	360	401	135	157	637	155	360	401	135	157
MAS [cycles]	46	46	28	28	44	30	46	46	28	28
Perf./slice ratio	**8.62**	**7.06**	**12.96**	**10.4**	**1.00**	**1.66**	–	–	–	–

Table II
COMPARISON OF IMPLEMENTATION RESULTS WITH VARIOUS ECC FPGA IMPLEMENTATIONS IN TERMS OF AREA, MAX. CLOCK FREQUENCY AND SPEED OF EXECUTION OF ELLIPTIC CURVE POINT MULTIPLICATION (ECMP).

Reference	**Area**			**Max. Freq.**	**ECPM Latency**	**FPGA**	**Remarks**
	# Logic	**# MULTs**	**# BRAMs**	**(MHz)**	**(ms)**		
This work: P-224 16-bit	773 Slices	1	3	210.0	8.20	Virtex-II Pro	NIST Curve P-224
P-256 16-bit	773 Slices	1	3	210.0	10.02	Virtex-II Pro	NIST Curve P256
P-224 32-bit	1158 Slices	4	3	210.0	3.64	Virtex-II Pro	NIST Curve P-224
P-256 32-bit	1158 Slices	4	3	210.0	4.52	Virtex-II Pro	NIST Curve P-256
[9] 16-bit	1832 Slices	2	9	108.20	29.83	Virtex-II Pro	any prime curve
[9] 32-bit	2085 Slices	7	9	68.17	15.76	Virtex-II Pro	any prime curve
[18] P-224	1580 Slices	26 (DSP)	11	487	0.45	Virtex-4	NIST Curve P-224
[18] P-256	1715 Slices	32 (DSP)	11	490	0.62	Virtex-4	NIST Curve P-256
[19] P-256	15755 Slices	256	0	39.5	3.84	Virtex-II Pro	NIST Curve P-256
[8] P-192	11416 LUTs 5735 FFs	0	35	40.0	3.0	Xilinx XCV1000	NIST Curve P-192
[20]	27597 Slices	0	0	40	17.7	Spartan-3S-5000	with RSA support
[21]	5614 Slices	0	0	91.31	42.52	Virtex-1000E	160-bit ECC
[22]	20000 Slices	0	0	200	1.66	XC5VLX30	any prime curve
[23] design 1	13661 Slices	0	0	43	9.2	Virtex-4	any 256-bit curve
[23] design 2	20123 Slices	0	0	43	7.7	Virtex-4	any 256-bit curve
[24]	9177 ALM	96	0	157.2	0.68	Stratix II	any 256-bit curve

mately two times less slices and hardware multipliers and three times less BRAMs. Putting it all together, our core is 6-7 times more efficient than the smallest ECC FPGA implementation of [9] using the simple *performance per slice* metric. The critical path appears in the control FSMs, therefore in our generic implementation the maximum clock frequency for both 16-bit and 32-bit is identical. When optimizing the FSMs for a single architecture we can achieve 265 MHz, 225 MHz, 150 MHz and 120 MHz performance for Xilinx 16-bit, Xilinx 32-bit, Microsemi 16-bit and Microsemi 32-bit, respectively. We focus at SPA and DPA evaluation within future work.

VII. CONCLUSION

We presented the MicroECC processor designed to perform ECC computations on standardized elliptic curves over prime fields. The modular arithmetic is developed as a specific hardware component, while all ECC operations are handled by a embedded program executed on a specifically designed processor. In contrast to [9], we used a fast reduction algorithm to perform the modular multiplication instead of using Montgomery's multiplication method. Moreover, we replace all 256-bit registers for intermediate results with storage provided by two dual-port memories. Comparing our W-bit core configuration to handle 256-bit operands, we could triple the performance for a point multiplication while simultaneously saving more than approximately 50% of hardware resources compared to [9]. The implemented point multiplication embedded program consumes just one quarter of the program memory capacity and one half of the data memory capacity. Note that the majority of the data memory is used for storing constants (curve parameters, multiples of primes and commands for the fast reduction algorithm control). Thus there is sufficient space in both memories to implement high-level ECC protocols and algorithms without any extra hardware cost.

Algorithm 2 Modular multiplication using W-bit words and 256-bit operands. The clk represents increments of clock cycles.

```
Require: operands $a, b \in [0, p)$, where $p =$ P-256
Ensure: $c = a \times b \bmod$ P-256.
 1: compute integer mult. using operand scan method:
 2: $R \leftarrow 0, \quad clk \leftarrow 0$
 3: for $k = 0$ to $2N - 2$ do
 4:     for each element $(i, j) | i + j = k, 0 \leq i, j \leq N - 1$ do
 5:         $R \leftarrow a_i \times b_i + R, \quad clk \leftarrow clk + 1$
 6:     end for
 7:     $c_i \leftarrow R_0, \quad R \leftarrow R >> W$
 8: end for
 9: $clk \leftarrow clk + 1, \quad c_{2N-1} \leftarrow R_0$
10: compute FR: $acc = 0, \quad k \leftarrow 0, \quad j \leftarrow 0$
11: load FR command: $cmd_k \leftarrow$ DM
12: extract indexes $i$ and $ii$ of $c$ from $cmd_k, \quad k \leftarrow k + 1$
13: repeat
14:     repeat
15:         $acc \leftarrow acc + c_i \pm c_{ii}$
16:         load FR command: $cmd_k \leftarrow$ DM
17:         extract index $i$ of $c$ from $cmd_k$
18:         $k \leftarrow k + 1, \quad clk \leftarrow clk + 1$
19:     until $cmd_k =$ “The $r_j$ is complete”
20:     $r_j \leftarrow acc \bmod 2^W$
21:     $acc \leftarrow acc >> W$
22:     $j \leftarrow j + 1,$
23: until $cmd_k =$“FR is complete”
24: compute final correction: $clk \leftarrow clk + 1$
25: according to the value of $a$ determine multiple $X$ of P-256 to be added/subtracted.
26: $a = X \times$ P-256, $\quad b = r$
27: call algorithm for modular addition/subtraction
28: return $r = c$
```

References

[1] V. Miller, “Uses of Elliptic Curves in Cryptography,” in *Advances in Cryptology – Proceedings of CRYPTO 1985*, ser. LNCS, H. C. Williams, Ed., vol. 218. Springer, 1986, pp. 417–426.

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

[3] A. K. Lenstra and E. R. Verheul, “Selecting Cryptographic Key Sizes,” *Journal of Cryptology*, vol. 14, no. 4, pp. 255–293, 2001.

[4] *IEEE P1363 Standard Specifications for Public Key Cryptography*, Institute of Electrical and Electronics Engineers, 2000.

[5] ANSI X9.62-2005, “American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA),” http://www.x9.org, 2005.

[6] Certicom Research, “Standards for Efficient Cryptography – SEC 1: Elliptic Curve Cryptography,” Available at http://www.secg.org/secg_docs.htm, May 2009, version 2.0.

[7] ——, “Standards for Efficient Cryptography – SEC 2: Recommended Elliptic Curve Domain Parameters,” Available at http://www.secg.org/secg_docs.htm, January 2010, version 2.0.

[8] G. Orlando and C. Paar, “A Scalable $GF(p)$ Elliptic Curve Processor Architecture for Programmable Hardware,” in *CHES*, ser. LNCS, Ç. K. Koç, D. Naccache, and C. Paar, Eds., vol. 2162. Springer, 2001, pp. 356–371.

[9] J. Vliegen, N. Mentens, J. Genoe, A. Braeken, S. Kubera, A. Touhafi, and I. Verbauwhede, “A compact FPGA-based architecture for elliptic curve cryptography over prime fields,” in *ASAP*, July 2010, pp. 313–316.

[10] P. L. Montgomery, “Speeding the Pollard and elliptic curve methods of factorization,” *Mathematics of Computation*, vol. 48, no. 177, pp. 243–264, 1987.

[11] P. H. W. Leong and I. K. H. Leung, “A microcoded elliptic curve processor using FPGA technology,” *IEEE Trans. VLSI Syst.*, vol. 10, no. 5, pp. 550–559, 2002.

[12] T. Güneysu and C. Paar, “Ultra High Performance ECC over NIST Primes on Commercial FPGAs,” in *CHES*, ser. Lecture Notes in Computer Science, vol. 5154. Springer, 2008, pp. 62–78.

[13] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. New York: Springer, 2004.

[14] J. A. Solinas, “Generalized Mersenne Numbers,” National Security Agency (NSA), Tech. Rep., Sep. 1999.

[15] National Institute of Standards and Technology (NIST), “Recommended Elliptic Curves for Federal Government Use,” July 1999.

[16] P. Kocher, J. Jaffe, and B. Jun, “Introduction to Differential Power Analysis and Related Attacks,” Cryptographic Research Inc., 1998, 1998, http://www.cryptography.com/dpa/technical.

[17] J. Fan, X. Guo, E. De Mulder, P. Schaumont, B. Preneel, and I. Verbauwhede, “State-of-the-art of secure ECC implementations: a survey on known side-channel attacks and countermeasures,” in *HOST*, IEEE, Ed., 2010, pp. 76–87.

[18] Güneysu, T. and Paar, C., “Ultra High Performance ECC over NIST Primes on Commercial FPGAs,” in *CHES*, 2008, pp. 62–78.

[19] C. McIvor, M. McLoone, and J. McCanny, “An FPGA Elliptic Curve Cryptographic accelerator over GF(p),” in *Irish Signals and Systems Conference (ISSC)*, 2004, pp. 589–594.

[20] K. Sakiyama, N. Mentens, L. Batina, B. Preneel, and I. Verbauwhede, “Reconfigurable modular arithmetic logic unit supporting high-performance RSA and ECC over GF(p),” in *International Journal of Electronics*, 2007, pp. 501–514.

[21] S. Örs, L. Batina, B. Preneel, and J. Vandewalle, “Hardware Implementation of an Elliptic Curve Processor over GF(p),” in *International Journal of Embedded Systems*, 2002, pp. 433–443.

[22] L. Tawalbeh, A. Mohammad, and A. Gutub, “Efficient FPGA Implementation of a Programmable Architecture for GF(p) Elliptic Curve Crypto Computations,” *Journal of Signal Processing Systems*, vol. 59, pp. 233–244, 2010, 10.1007/s11265-009-0376-x.

[23] S. Ghosh, M. Alam, D. R. Chowdhury, and I. S. Gupta, “Parallel crypto-devices for GF(p) elliptic curve multiplication resistant against side channel attacks,” *Comput. Electr. Eng.*, vol. 35, pp. 329–338, March 2009.

[24] N. Guillermin, “A high speed coprocessor for elliptic curve scalar multiplications over F_p,” in *CHES*, ser. LNCS, vol. 6225, 2010, pp. 48–64.