Ultra High Performance ECC over NIST Primes on Commercial FPGAs

Tim Güneysu and Christof Paar

Horst Görtz Institute for IT Security, Ruhr University Bochum, Germany
{gueneysu,cpaar}@crypto.rub.de

Abstract. Elliptic Curve Cryptosystems (ECC) have gained increasing acceptance in practice due to their significantly smaller bit size of the operands compared to other public-key cryptosystems. Since their computational complexity is often lower than in the case of RSA or discrete logarithm schemes, ECC are often chosen for high performance public-key applications. However, despite a wealth of research regarding high-speed software and high-speed FPGA implementation of ECC since the mid 1990s, providing truly high-performance ECC on readily available (i.e., non-ASIC) platforms remains an open challenge. This holds especially for ECC over prime fields, which are often preferred over binary fields due to standards in Europe and the US.

This work presents a new architecture for an FPGA-based ultra high performance ECC implementation over prime fields. Our architecture makes intensive use of the DSP blocks in modern FPGAs, which are embedded arithmetic units actually intended to accelerate digital signal processing algorithms. We describe a novel architecture and algorithms for performing ECC arithmetic and describe the actual implementation of standard compliant ECC based on the NIST primes P-224 and P-256. We show that ECC on Xilinx's Virtex-4 SX55 FPGA can be performed at a rate of more than 37,000 point multiplications per second. Our architecture outperforms all single-chip hardware implementations over prime fields in the open literature by a wide margin.

Keywords: Elliptic Curve Cryptosystems, FPGA, High-Performance.

1 Introduction

With the explosive growth of Internet-based applications like ecommerce, peer-to-peer networks and distributed gaming as well as embedded ones — ranging from mobile over set-top boxes to automotive — the demand for security in such systems has also grown dramatically. In these applications, asymmetric cryptography is used to achieve a large variety of security goals. However, asymmetric cryptographic algorithms are extremely arithmetic intensive since their security assumptions rely on computational problems which are considered to be hard in combination with parameters of significant bit sizes.

Neal Koblitz and Victor Miller proposed independently in 1985 [20,17] the use of Elliptic Curve Cryptography providing similar security compared to classical cryptosystems but using smaller keys. This benefit allows for greater efficiency

E. Oswald and P. Rohatgi (Eds.): CHES 2008, LNCS 5154, pp. 62–78, 2008.

when using ECC (160–256 bit) compared to RSA or discrete logarithm schemes over finite fields (1024–4096 bit) while providing an equivalent level of security [18]. Due to this, ECC has become the most promising candidate for many new applications, especially in the embedded domain, which is also reflected by several standards by IEEE, ANSI and SECG [15,1,5,6].

In addition to many new "lightweight" applications (e.g., digital signature on RFID-like devices), there are also many new applications which call for high-performance asymmetric primitives. Even though very fast public-key algorithms can be provided for PC and server applications by accelerator cards equipped with ASICs, providing very high speed solutions in embedded devices is still a major challenge. Somewhat surprisingly, there appears to be extremely few, if any, commercially available ASICs or chip sets that provide high speed ECC and which are readily available for integration in general embedded systems. A potential alternative is provided by Field Programmable Gate Arrays (FPGA). FPGAs have evolved over the last decade to a powerful alternative for classical ASIC circuits. In addition, FPGAs provide the advantage of dynamic and flexible circuit reconfigurability allowing for rapid prototyping at little development costs. However, despite a wealth of research regarding high-speed FPGA (and high-speed software) implementation of ECC since the mid 1990s, providing truly high-performance ECC (i.e., to reach less than 100μs per point multiplication) on readily available platforms remains an open challenge. This holds especially for ECC over prime fields, which are often preferred over binary fields due to standards in Europe and the US, and a somewhat clearer patent situation.

In this work, we propose a novel hardware architecture based on reconfigurable FPGAs supporting ECC cryptography over prime fields $GF(p)$ offering the highest single-chip performance reported in literature up to now. Usually, known ECC implementations for reconfigurable logic implement the computationally expensive low-level arithmetic in configurable logic elements, allowing for greatest flexibility but offering only moderate performance. Some implementations have attempted to address this problem by using dedicated arithmetic hardware in the reconfigurable device for specific parts of the computations, like built-in 18x18 multipliers. But other components of the circuitry for field addition, subtraction and inversion have been still implemented in the FPGA's fabric which usually leads to a significant decrease in performance.

The central idea of this contribution is to relocate the arithmetic intensive operations of ECC over prime fields *entirely* in dedicated hardcore units on the FPGA actually reserved for use in Digital Signal Processing (DSP) filter applications. These DSP accelerating functions are built-in components in the static logic of modern FPGA devices capable to perform integer multiplication, addition and subtraction as well as a multiply-accumulate operation.

2 Previous Work

We briefly summarize previously published results of relevance to this contribution. There is a wealth of publication addressing ECC hardware architectures,

and a good overview can be found in [8]. In the case of high speed architectures for ECC, most implementation primarily address elliptic curves over binary fields $GF(2^m)$ since the arithmetic is more hardware-friendly [22,10]. Our work, however, focuses on the prime field $GF(p)$. First implementations for ECC over prime fields $GF(p)$ have been proposed by [23,24] demonstrating ECC processors built completely in reconfigurable logic. The contribution by [19] proposes a high-speed ECC crypto core for arbitrary moduli with up to 256 bit length designed on a large number of built-in multiplier blocks of FPGA devices providing a significant speedup for modular multiplications. However, other field operations have been implemented in the FPGA fabric, resulting in a very large design (15,755 slices and 256 multiplier blocks) on a large Xilinx XC2VP125 device. The architecture presented in [7] was designed to achieve a better trade-off between performance and resource consumption. According to the contribution, an area consumption of only 1,854 slices and a maximum clock speed of 40 MHz can be achieved on a Xilinx Virtex-2 XC2V2000 FPGA for a parameter bit length of 160 bit.

Our approach to implementing an FPGA-based ECC engines was to shift *all* field operations into the integrated DSP building blocks available on modern FPGAs. We show that this approach leads to an extremely high throughput. Furthermore, our strategy frees most configurable logic elements on the FPGA for other applications and requires less power compared to a conventional design. To the best of our knowledge, this architecture offers the fastest performance for ECC computations over prime fields with up to 256 bit security in reconfigurable logic.

3 Mathematical Background

In the following, we will briefly introduce to the mathematical background relevant for this work. We will start with a short review of the Elliptic Curve Cryptosystems (ECC). Please note that only ECC over prime fields $GF(p)$ will be subject of this work since binary extensions fields $GF(2^m)$ require binary arithmetic which is not (yet) natively supported by DSP blocks.

3.1 Elliptic Curve Cryptography

Let p be a prime with $p > 3$ and $\mathbb{F}_p = GF(p)$ the Galois Field over p. Given the Weierstrass equation of an elliptic curve

$$\mathcal{E} : y^2 = x^3 + ax + b,$$

with $a, b \in GF(p)$ and $4a^3 + 27b^2 \neq 0$, points $\mathcal{P}_i \in \mathcal{E}$, we can compute tuples (x, y) also considered as points on this elliptic curve $\mathcal{E}$. Based on a group of points defined over this curve, ECC arithmetic defines the addition $\mathcal{R} = \mathcal{P} + \mathcal{Q}$ of two points $\mathcal{P}, \mathcal{Q}$ using the *tangent-and-chord* rule as the primary group operation. This group operation distinguishes the case for $\mathcal{P} = \mathcal{Q}$ (*point doubling*) and $\mathcal{P} \neq \mathcal{Q}$ (*point addition*). Furthermore, formulas for these operations vary

for affine and projective coordinate representations. Since affine coordinates require the availability of fast modular inversion, we will focus on projective point representation to avoid the implementation of a costly inversion circuit. Given two points $\mathcal{P}_1, \mathcal{P}_2$ with $\mathcal{P}_i = (X_i, Y_i, Z_i)$ and $\mathcal{P}_1 \neq \mathcal{P}_2$, the sum $\mathcal{P}_3 = \mathcal{P}_1 + \mathcal{P}_2$ is defined by

$$\begin{aligned} A &= Y_2Z_1 - Y_1Z_2 \quad & B &= A^2Z_1Z_2 - C^3 - 2C^2X_1Z_2 \quad & C &= X_2Z_1 - X_1Z_2 \\ X_3 &= BC & Y_3 &= A(C^2X_1Z_2 - B) - C^3Y_1Z_2 & Z_3 &= C^3Z_1Z_2, \end{aligned} \tag{1}$$

where A, B, C are auxiliary variables and $\mathcal{P}_3 = (X_3, Y_3, Z_3)$ is the resulting point in projective coordinates. Similarly, for $\mathcal{P}_1 = \mathcal{P}_2$ the point doubling $\mathcal{P}_3 = 2\mathcal{P}_1$ is defined by

$$\begin{aligned} A &= aZ^2 + 3X^2 \quad & B &= YZ \quad C = XYB & D &= A^2 - 8C \\ X_3 &= 2BD & Y_3 &= A(4C - D) - 8B^2Y^2 & Z_3 &= 8B^3. \end{aligned} \tag{2}$$

Most ECC-based cryptosystems rely on the Elliptic Curve Discrete Logarithm Problem (ECDLP) and thus employ the technique of point multiplication $k \cdot \mathcal{P}$ as cryptographic primitive, i.e., a k times repeated point addition of a base point $\mathcal{P}$. Precisely, the ECDLP is the fundamental cryptographic problem used in protocols and crypto schemes like the Elliptic Curve Diffie-Hellman key exchange [9], the ElGamal encryption scheme [12] and the Elliptic Curve Digital Signature Algorithm (ECDSA) [1].

3.2 Standardized General Mersenne Primes

The arithmetic for ECC point multiplication is based on modular computations over a prime field $GF(p)$. These computations always include a subsequent step to reduce the result to the domain of the underlying field. Since the reduction is very costly for general primes due to the demand for a multi-precision division, special primes have been proposed by Solinas [26] which have been finally standardized in [21]. These primes provide efficient reduction algorithms based on a sequence of multi-precision addition and subtractions only and eliminate the need for the costly division. Special primes P-l with bitlengths $l = \{192, 224, 256, 384, 521\}$ are part of the standard. But we believe that the primes P-224 and P-256 are the most relevant bit sizes for future implementations of the next decades.

According to Algorithm 1 the modular reduction for P-224 can be performed with two 224-bit subtractions and additions. However, these four consecutive operations can lead to a potential over- and underflow in step 1. With $Z = z_1 + z_2 + z_3 - z_4 - z_5$, we can determine the bounds $-2p < Z < 3p$ reducing the number of final correction steps to two additions or subtractions to compute the correctly bounded $c \bmod p_{224}$.

Algorithm 2 presents the modular reduction for P-256 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 result to be corrected is $-4p < Z < 5p$.

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

Input: Double-sized integer $c = (c_{13}, \ldots, c_2, c_1, c_0)$ in base 2^{32} and $0 \geq c \geq$ P-224^2
Output: Single-sized integer c mod P-224.
1: Concatenate c_i to following 224-bit integers z_j:

$$z_1 = (c_6, c_5, c_4, c_3, c_2, c_1, c_0),\ z_2 = (c_{10}, c_9, c_8, c_7, 0, 0, 0),$$
$$z_3 = (0, c_{13}, c_{12}, c_{11}, 0, 0, 0),\ z_4 = (0, 0, 0, 0, c_{13}, c_{12}, c_{11}),$$
$$z_5 = (c_{13}, c_{12}, c_{11}, c_{10}, c_9, c_8, c_7)$$

2: Compute $c = (z_1 + z_2 + z_3 - z_4 - z_5 \bmod \text{P-224})$

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

Input: Double-sized integer $c = (c_{15}, \ldots, c_2, c_1, c_0)$ in base 2^{32} and $0 \geq c \geq$ P-256^2
Output: Single-sized integer c mod 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 $c = (z_1 + 2z_2 + 2z_3 + z_4 + z_5 - z_6 - z_7 - z_8 - z_9 \bmod \text{P-256})$

4 An Efficient ECC Architecture Using DSP Cores

In this section we demonstrate how to implement ECC over NIST primes P-224 and P-256 using available DSP blocks of Xilinx Virtex-4 FPGAs.

4.1 DSP-Accelerator Blocks in FPGAs

Modern FPGA devices like Xilinx Virtex-4 and Virtex-5 as well as Altera Stratix FPGAs have been equipped with dedicated arithmetic hardcore extensions to accelerate, in particular, digital signal processing applications. These function blocks (*DSP blocks*) can be used to build a more efficient implementation in terms of performance *and* reduce at the same time the demand for logical elements. In general, DSP blocks of FPGAs can be programmed to perform basic arithmetic functions, especially, multiplication, addition and subtraction of (un)signed integers. A common DSP component comprises an l_M-bit signed integer multiplier coupled with an l_A-bit signed adder, where $l_A > l_M$ holds. For enabling maximum performance, the multiplier and adder block can be augmented with pipeline registers to reduce signal propagation delays between components. Using different data paths, DSP blocks can operate on external inputs A, B, C as well as on feedback values from accumulation or even results $P_{j\pm1}$ from a

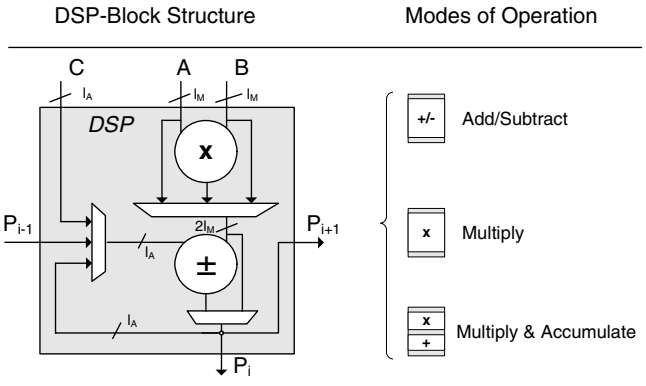

Fig. 1. Generic and simplified structure of DSP-blocks of advanced FPGA devices

neighboring DSP block. Figure 1 shows the generic DSP-block used in recent Xilinx FPGA devices [29].

4.2 ECC Engine Design Criteria

When using DSP blocks to develop a high-speed ECC design, there are several criteria which should be met to exploit their full performance. Note that the following aspects have been designed to target the requirements of Xilinx Virtex-4 FPGAs:

1. *Build DSP cascades:* Neighboring DSP blocks can be cascaded to widen or extent their atomic operand width (e.g., from 18 bit to 256 bit).
2. *Use DSP routing paths:* DSPs have been provided with inner routing paths connecting two adjacent blocks. It is advantageous in terms of performance to use these paths as frequently as possible instead of using FPGA's general switching matrix for connecting logic blocks.
3. *Consider DSP columns:* Within a Xilinx FPGA, DSPs are aligned in columns, i.e., routing paths between DSPs within the same column are efficient while a switch in columns can lead to degraded performance. Hence, DSP cascades should not exceed the column width (typically 32/48/64 DSPs per column).
4. *Use DSP pipeline registers:* DSP blocks feature pipeline stages which should be used to achieve the maximum clock frequency supported by the device (up to 500 MHz).
5. *Use different clock domains:* Optimally, DSP blocks can be operated at maximum device frequency. This is not necessarily true for the remainder of the design so that separate clock domains should be introduced (e.g. by halving the clock frequency for control signals) to address the critical paths in each domain individually.

4.3 Arithmetic Units

According to the EC arithmetic introduced in Section 3.1, an ECC engine over $GF(p)$ based on projective coordinates requires functionality for modular addition, subtraction and multiplication. Since modular addition and subtraction is very similar, both operation are combined. In the following description we will assume a Virtex-4 FPGA as reference device and corresponding DSP block arithmetic with word sizes $l_A = 32$ and $l_M = 16$ for unsigned addition and multiplication, respectively. Note that native support by the DSP blocks on a Virtex-4 device is available for up to 48-bit signed addition and 18-bit signed multiplication.

Modular Addition/Subtraction. Let $A, B \in GF(P)$ be two multi-precision operands with lengths $|A|, |B| \leq l$ and $l = \lfloor \log_2 P \rfloor + 1$. Modular addition $C = A+B \bmod P$ and subtraction $C = A-B \bmod P$ can be efficiently computed according to Algorithm 3:

Algorithm 3. Modular addition and subtraction

Input: A, B, P with $0 \leq A, B < P$;
Operation flag $f \in \{0, 1\}$ denotes a subtraction when $f = 1$ and addition otherwise
Output: $C = A \pm B \bmod P$
1: $(C_{\mathsf{IN0}}, S_0) = A + (-1)^f B$;
2: $(C_{\mathsf{IN1}}, S_1) = S_0 + (-1)^{1-f} P$;
3: Return $S_{|f-C_f|}$;

For using DSP blocks, we need to divide the l-bits operands into multiple words each having a maximum size of l_A bit due to the limited width of the DSP input port. Thus, all inputs A, B and P to the DSP blocks can be represented in the form $X = \sum_{i=0}^{n_A-1} x_i \cdot 2^{i \cdot l_A}$, where $n_A = \lceil l/l_A \rceil$ denotes the number of words of an operand. According to Algorithm 3, we employ two cascaded DSP blocks, one for computing $s_{(0,i)} = a_i \pm (b_i + C_{\mathsf{IN0}})$ and a second for $s_{(1,i)} = s_{(0,i)} \mp (p_i + C_{\mathsf{IN1}})$. The resulting values $s_{(0,i)}$ and $s_{(1,i)}$ each of size $|s_{(j,i)}| \leq l_A + 1$ are temporarily stored and recombined to S_0 and S_1 using shift registers (SR). Finally, a 2-to-1 l-bit output multiplexer selects the appropriate value $C = S_i$. Figure 2 presents a schematic overview of a combined modular addition and subtraction based on two DSP blocks. Note that DSP blocks on Virtex-4 FPGAs provide a dedicated carry input c_{IN} but *no* carry output c_{OUT}. Particularly, this fact requires extra logic to compensate for duplicate carry propagation to the second DSP which is due to the fixed cascaded routing path between the DSP blocks. In this architecture, each carry is considered twice, namely in $s_{0,i+1}$ and $s_{1,i}$ what needs to be corrected. This special carry treatment requires a wait cycle to be introduced so that one l_A-bit word can be processed each two clock cycles. However, this is no restriction for our architecture since we design for *parallel* addition and multiplication so that the (shorter) runtime of an addition is completely hidden in the duration of a concurrent multiplication operation.

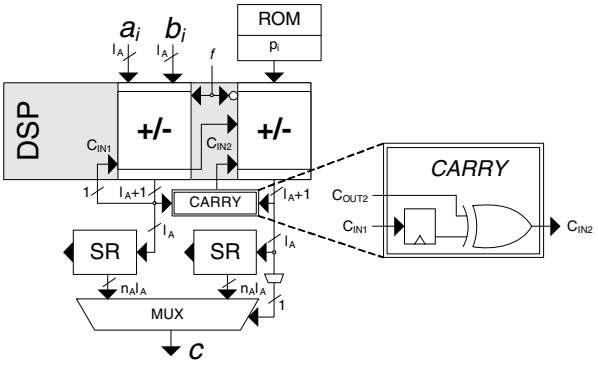

Fig. 2. Modular addition/subtraction based on DSP-blocks

Modular Multiplication. The most straightforward multiplication algorithm to implement the multiplication with subsequent NIST prime reduction (cf. Section 3.2) is the schoolbook multiplication method with a time complexity of $\mathcal{O}(n^2)$ for n-bit inputs. Other methods, like the Karatsuba algorithm [16], trade multiplications for additions using a divide-and-conquer approach. Due to the higher number of additions, this latter strategy is only preferable in case that the complexity costs of an an addition is significantly below that of a multiplication [28]. But even when neglecting any further control overhead introduced by the Karatsuba method, this does not hold for Virtex-4 devices since multiplication is comparably cheap within the DSP blocks. Let $A, B \in GF(P)$ be two multi-precision integers with bit length $l \leq \lfloor \log_2 P \rfloor + 1$. According to the limited input size l_M of DSP blocks, we split now the values A, B in $n_M = \lceil l/l_M \rceil$ words represented as $X = \sum_{i=0}^{n_M-1} x_i \cdot 2^{il_M}$. Schoolbook multiplication computes $C = A \cdot B$ based on accumulation of $(n_M)^2$ products $C = \sum_{i=0}^{2n_M} 2^{i \cdot n_M} \sum_{j=0}^{i} a_j b_{i-j}$ providing a result C of size $|C| \leq 2n_M$. For parallel execution on n_M DSP units, we compacted the order of inner product computations as shown in Figure 3. All n_M DSP blocks operate in a loadable *Multiply-and-Accumulate* mode (MACC) so that intermediate results remain in the corresponding DSP block until an inner product $s_i = \sum_{j=0}^{i} a_j b_{i-j}$ is fully computed. Note that s_i returned from the n_M DSP blocks are not aligned and can vary in size up to $|s_i| \leq 2l_M + \log_2(n_M) = l_{ACC} = 36$ bits. Thus, all s_i need to be converted to non-redundant representation to finally form the final product of words c_i with maximum size $2l_M$ each. Hence, we feed all values into a subsequent accumulator to combine each s_i with the corresponding bits of s_{i-1} and s_{i+1}. Considering the special input constraints, timing conventions and carry transitions of DSP blocks, we developed Algorithm 4 to address the accumulation of inner products based on two DSP blocks performing l_{ACC}-bit additions.

Figure 4 gives a schematic overview of the multiplication circuit returning the full-size product C. This result has to be reduced using the fast NIST prime reduction scheme discussed in the next section.

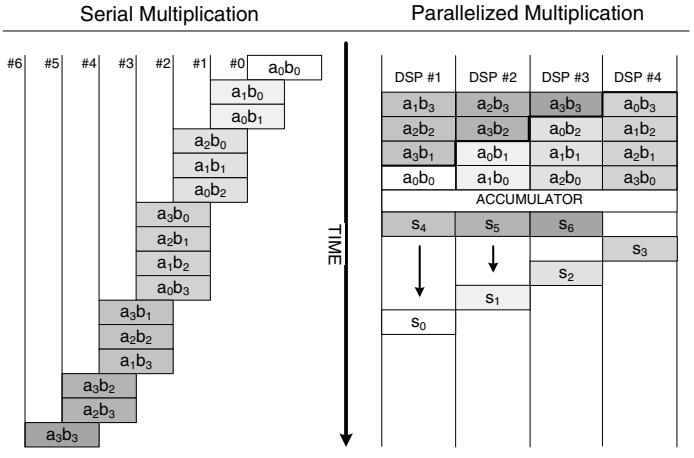

Fig. 3. Parallelizing multiplication for efficient DSP-based computation

Algorithm 4. Accumulation of partial product c_i

Input: Partial products s_i with bitsize $|s_i| \leq l_{ACC}$ for $i = 0 \ldots 2n_M - 1$ and $l_{ACC} = 2l_M + \log_2(n_M)$
Output: Product $C = (c_{2n_M}, \ldots, c_0)$ with bitsize $|C| \leq 2l$
1: $s_{(-1)} \rightarrow 0$; $c_{(-1)} \rightarrow 0$
2: **for** $i = 0$ to $2n_M - 2$ by 2 **do**
3: $d_i \rightarrow \text{ADD}(s_{i-1}[l_{ACC} - 1 \ldots l_M], s_i[l_{ACC} \ldots 0])$
4: $c_i \rightarrow \text{ADD}(d_i[l_{ACC} \ldots l_M], (s_{i+1}[l_M \ldots 0] | c_{i-1}[3l_M \ldots 2l_M]))$
5: **end for**
6: **return** $c = (c_{2n_M-1}, \ldots, c_0)$

Modular Reduction. At this point we will discuss the subsequent modular reduction of the $2n_M$-bit multiplication result C using the NIST reduction scheme. All fast NIST reduction algorithms rely on a reduction step (1) defined as a series multi-precision additions and subtractions followed by a correction step (2) to achieve a final value in the interval $[0, \ldots, P-1]$ (cf. Algorithms 1 and 2). To implement (1), we decided to use one DSP-block for each individual addition or subtraction, e.g., for the P-256 reduction we reserved a cascade of 8 DSP blocks. Each DSP performs one addition or subtraction and stores the result in a register whose output is taken as input to the neighboring block (data pipeline).

For the correction step (2), we need to determine *in advance* the possible overflow or underflow of the result returned by (1) to avoid wait or idle cycles in the pipeline. Hence, we introduced a Look-Ahead Logic (LAL) consisting of a separate DSP block which exclusively computes the expected overflow or underflow. Then, the output of the LAL is used to select a corresponding reduction value which are stored as multiple $\{0, \ldots, 5P\}$ in a ROM table. The ROM values are added or subtracted to the result of (1) by a sequence of two DSP blocks

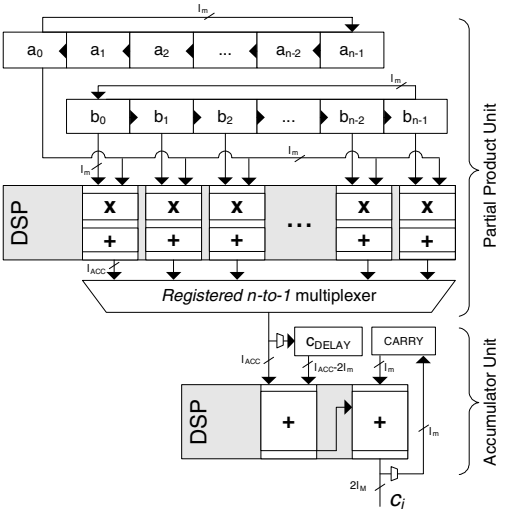

Fig. 4. An l-bit multiplication circuit employing a cascade of parallelly operating DSP blocks

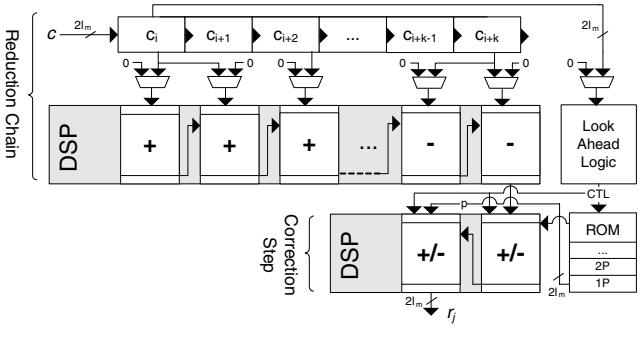

Fig. 5. Modular reduction for NIST-P-224 and P-256 using DSP blocks

ensuring that the final result is always in $\{0, \ldots, P-1\}$. Figure 5 depicts the general structure of the reduction circuit which is applicable for both primes P-224 and P-256.

4.4 ECC Core Architecture

With the basic field operations for $l-bit$ computations at hand supporting NIST primes P-224 and P-256, we have combined a modular multiplier and a modular subtraction/addition component with dual-port RAM modules (BRAM) and a state machine to build an ECC core. We have implemented an asymmetric datapath supporting two different operand lengths: the first operand provides full l-bit of data whereas the second operand is limited to 32-bit words so that

several words need to be transferred serially to generate the full l-bit input. This approach allows for direct memory accesses of our *serial-to-parallel* multiplier architecture. Note further that we introduced different clock domains for the core arithmetic based on the DSP blocks and the state machines for upper layers (running at half clock frequency only). An overview of the entire ECC core is shown in Figure 6. We implemented ECC group operations based on projective Chudnowsky coordinates[1] since the implementation should support to compute a point multiplication $k \cdot \mathcal{P}$ *as well as* a corresponding linear combination $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ based on a fixed base point $\mathcal{P} \in \mathcal{E}$, $k, r \in \{1, \ldots, ord(\mathcal{P}) - 1\}$ and $\mathcal{Q} \in \langle \mathcal{P} \rangle$. Both operations can be considered as basic ECC primitives, e.g., used for ECDSA signature generation and verification [1]. The computation of $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ can make use of *Shamir's trick* to efficiently compute several point products simultaneously [12]. For this first implementation of the point multiplication and the sake of simplicity, we used a standard double-and-add (binary method) algorithm [14], but more efficient windowing methods [2] can also be implemented without significantly increasing the resource consumption.

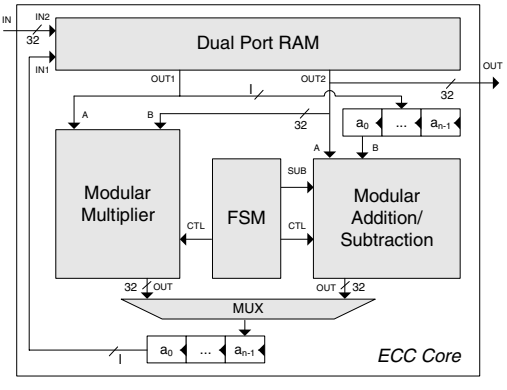

Fig. 6. Schematic overview of a single ECC core

4.5 ECC Core Parallism

Due the intensive use of DSP blocks to implement the core functionality of ECC, the resulting implementation requires only few reconfigurable logic elements on the FPGA. This allows for efficient multiple-core implementations on a single FPGA improving the overall system throughput by a linear factor n dependent on the number of cores. Note that most other high-performance implementations occupy the full FPGA due to their immense resource consumption so that these cannot easily be instantiated several times.

[1] ECC operations based on mixed affine-Jacobian coordinates are more efficient but more complex in hardware when considering precomputed points in Jacobian coordinates required for computing $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ as required for ECDSA signature verification.

Based on our synthesis results, the limiting factor of our architecture is the number of available DSP blocks of a specific FPGA device (cf. Section 5).

5 Implementation

The proposed architecture has been synthesized and implemented for the smallest available Xilinx Virtex-4 device (XC4VFX12-12SF363) and the corresponding results are presented in Subsection 5.1. This FPGA type offers 5,472 slices (12,288 4-input LUTs and flip flops) of reconfigurable logic, 32 DSP blocks and can be operated at a maximum clock frequency of 500 MHz. Furthermore, to demonstrate how many ECC computations can be performed using ECC core parallelism, we take a second device, the large Xilinx Virtex-4 XC4VSX55-12FF1148 providing the maximum number of 512 DSP blocks and 24,576 slices (49,152 4-input LUTs and flip flops) as a reference for a multi-core architecture.

5.1 Implementation Results

Based on the Post-Place and Route (PAR) results using Xilinx ISE 9.1 we can present the following performance and area details for ECC cores for primes P-224 and P-256 on the small XC4VFX12 device as shown in Table 1. Note that up to now the implementation for P-224 is not yet fully verified in functionality or optimized. The core for P-256, however, is already available for use in real-world products.

Table 1. Requirements and clock frequency of a single ECC core on a Virtex-4 FX 12 after PAR

Aspect	**ECC Core** P-224	**ECC Core** P-256
Slices occupied	1,580 (29%)	1,715 (31%)
4-input LUTs	1,825	2,589
Flip flops	1,892	2,028
DSP blocks	26	32
BRAMs	11	11
Frequency/Max. delay	487 MHz/2.050 ns	490 MHz/2.040 ns

5.2 Throughput of a Single ECC Core

Given an ECC core with a separate adder/subtracter and multiplier unit, we can perform a field multiplication and field addition simultaneously. By optimizing the execution order of the basic field operations, it is possible to perform all additions/subtraction required for the ECC group operation in parallel to a multiplication. Based on the runtimes of a single field multiplication, we can determine the number of required clock cycles for the operations $k \cdot \mathcal{P}$ and $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ using the implemented Double-and-Add algorithm. Moreover, we also give

Table 2. Performance of ECC operations based on a single ECC core using projective Chudnowsky coordinates on a Virtex-4 XC4VFX12 (Figures denoted with an asterisk are estimates)

Aspect	**ECC Core** P-224	**ECC Core** P-256
Cycles per MUL in $GF(p)$	58	70
Cycles per ADD/SUB in $GF(p)$	16	18
Cycles per ECC Addition (Chudnovsky)	812	980
Cycles per ECC Doubling (Chudnovsky)	580	700
Cycles $k \cdot \mathcal{P}$ (Double&Add)	219,878	303,450
Cycles $k \cdot \mathcal{P}$ (Window)	178,000*	243,000*
Cycles $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ (Double&Add)	265,959	366,905
Cycles $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ (Window)	194,000*	264,000*
Time and OP/s for $k \cdot \mathcal{P}$ (Double&Add)	452 μs/2214	620 μs/1614
Time and OP/s for $k \cdot \mathcal{P}$ (Window)	365 μs*/2740*	495 μs*/2020*
Time and OP/s for $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ (Double&Add)	546 μs/1831	749 μs/1335
Time and OP/s for $k \cdot \mathcal{P} + r \cdot \mathcal{Q}$ (Window)	398 μs*/2510*	540 μs*/1850*

estimates concerning their performance when using a window-based method [2] based on a window size $w = 4$.

Note that the specified timing considers signal propagation after complete PAR excluding the timing constraints from I/O pins since no underlying data communication layer was implemented. Hence, when being combined with an I/O protocol of a real-world application, the clock frequency will be slightly lower than specified in Table 1 and 3.

5.3 Multi-core Architecture

Since a single ECC core has obviously moderate resource requirements, it is possible to place multiple instances of the core on a larger FPGA. On a *single* XC4VSX55 device, we can implement, depending on the underlying prime field, between 16–18 ECC cores running in parallel (cf. Table 3). Due the small amount of LUTs and flip flops required for a single core, the number of available DSP blocks (and routing resources) on the FPGA is here the limiting factor.

5.4 Comparison

Based on our architecture, we can estimate a throughput of more than 37,000 point multiplications on the standardized elliptic curve P-224 per second which exceeds the throughput of all *single-chip* hardware implementation known to the authors by far. A detailed comparison with other implementations is presented in Table 4.

Table 3. PAR-Results for a multi-core architecture on a Virtex-4 XC4VSX55 device for ECC over prime fields P-224 and P-256 (Figures denoted with an asterisk are estimates)

Aspect	**ECC** P-224	**ECC** P-256
Number of Cores	18	16
Slices occupied	24,452 (99%)	24,574 (99%)
4-input LUTs	32,688	34,896
Flip flops	34,166	32,430
DSP blocks	468	512
BRAMs	198	176
Frequency/Max. delay	372 MHz/2.685 ns	375 MHz/2.665 ns
OP/s $k \cdot P$ (Double&Add)	30,438	19,760
OP/s $k \cdot P$ (Window)	37,700*	24,700*
OP/s $k \cdot P + r \cdot Q$ (Double&Add)	25,164	16,352
OP/s $k \cdot P + r \cdot Q$ (Window)	34,500*	22,700*

At this point we like to point out that the field of highly efficient *prime field* arithmetic is believed to be predominated by implementations on general purpose microprocessors rather than on FPGAs. Hence, we will also compare our hardware implementation against the performance of software solutions on recent microprocessors. Since most performance figures for software implementations are given in cycles rather than absolute times, we assumed for comparing throughputs that, on a modern microprocessor, repeated computations can be performed *without* interruption simultaneously on *all* available cores with no further cycles spent, e.g., on scheduling or other administrative tasks. Note that this is indeed a very optimistic assumption possibly overrating the performance of software implementations with respect to actual applications.

For example, a point multiplication using the highly efficient software implementation by Dan Bernstein based on floating point arithmetic for ECC over P-224 requires 839.000 cycles on an (outdated) Intel Pentium 4 [3] at 1.4GHz. According to our assumption for cycle count interpretation, this correlates to 1670 point multiplication per second.

Despite the good performance figures on this platform, we prefer to take more recent results, e.g., obtained from ECRYPT's eBATS project. According to the report from March 2007 [11], an Intel Core2 Duo running at 2.13 GHz is able to generate 1868 and 1494 ECDSA signatures based on the OpenSSL implementation for P-224 and P-256, respectively. Taking latest Intel Core2 Quad microprocessors into account, these performance figures might even double. We also compare our work to the very fast software implementation by [13] using an Intel Core2 system at 2.66 GHz. However, in this contribution the special Montgomery and non-standard curve over $\mathbb{F}_{2^{255}-19}$ is used instead of a standardized NIST prime. Despite of that, for the design based on this curve the authors report the impressive throughput of 6700 point multiplications per second.

Table 4. Selected high-performance implementations of public-key cryptosystems

Scheme	Device	Implementation	Logic	Clock	Time
This work	XC4VFX12-12	GF(p), NIST-224	1580 LS/26 DSP	487 MHz	365 μs
	XC4VFX12-12	GF(p), NIST-256	1715 LS/32 DSP	490 MHz	495 μs
	XC4VSX55-12	GF(p), NIST-224	24452 LS/468 DSP	372 MHz	26.5 μs
	XC4VSX55-12	GF(p), NIST-256	24574 LS/512 DSP	375 MHz	40.5 μs
ECC [23]	XCV1000E	GF(p), NIST-192	5708 LS	40 MHz	3 ms
ECC [19]	XC2VP125-7	GF(p), 256-bit	15755 LS/256 MUL	39.5 MHz	3.84 ms
ECC [24]	0.13 μm CMOS	GF(p), 160-bit	117500 GE	137.7 MHz	1.21 ms
ECC [3]	Intel Pentium4	GF(p), NIST-224	32 bit μP	1.4 GHz	599 μs
ECC [11]	Intel Core2 Duo	GF(p), NIST-256	64 bit μP	2.13 GHz	669[a] μs
ECC [13]	Intel Core2 Duo	GF($2^{255} - 19$)	64 bit μP	2.66 GHz	145 μs
RSA[4]	XC40250XV	1024-bit	6826 CLB	45.2 MHz	3.1 ms
RSA[27]	XC4VFX12-10	1024-bit (DSP)	3937 LS/17 DSP	400 MHz	1.71 ms
RSA[25]	0.5 μm CMOS	1024-bit	28,000 GE	64 MHz	46 ms

[a] Note that this figure reflects a full ECDSA signature generation rather than a point multiplication.

For a fair comparison with software solutions it should be considered that a single Virtex-4 SX 55 costs about US$ 1,170[2]. Recent microprocessors like the Intel Core2 Duo, however, are available at only about a quarter of that price. With this in mind, we might not be able to beat all software implementation in terms of the cost-performance ratio, but we still like to point out that our FPGA-based design - as the fastest reported hardware implementation so far - definitely closes the performance gap between software and hardware implementations for ECC over prime fields. Furthermore, we like to emphasize again that all software related performance figures are based on very optimistic assumptions.

6 Conclusion

We presented a novel ECC implementation for fields over NIST primes P-224 and P-256. Due to the exhaustive utilization of DSP blocks, which are contained as hardcores in modern FPGA devices, we are able to perform the critical components computing low-level integer arithmetic operations nearly at maximum device frequency. Furthermore, considering a multi-core architecture on a Virtex-4 XC4VSX55 FPGA, we can achieve a throughput of more than 24,000 and 37,000 point multiplications per second for P-256 and P-224, respectively, what significantly exceeds the performance of all other hardware implementation known to the authors and comes close to the cost-performance ratio provided by the fastest available software implementations in the open literature.

[2] Market price for a single device in May 2008.

References

1. ANSI X9.62-2005. American National Standard X9.62: The Elliptic Curve Digital Signature Algorithm (ECDSA). Technical report, Accredited Standards Committee X9 (2005), http://www.x9.org
2. Avanzi, R.M., Cohen, H., Doche, C., Frey, G., Lange, T., Nguyen, K., Vercauteren, F.: Handbook of Elliptic and Hyperelliptic Curve Cryptography. Chapman & Hall/CRC (2005)
3. Bernstein, D.J.: A software implementation of NIST P-224. In: Presentation at the 5th Workshop on Elliptic Curve Cryptography (ECC 2001), October 29-31, 2001, University of Waterloo (2001), http://cr.yp.to/nistp224/timings.html
4. Blum, T., Paar, C.: High radix Montgomery modular exponentiation on reconfigurable hardware. IEEE Transactions on Computers 50(7), 759–764 (2001)
5. Certicom research. Standards for Efficient Cryptography — SEC 1: Elliptic Curve Cryptography. Version 1.0 (September 2000), http://www.secg.org/secg_docs.htm
6. Certicom research. Standards for Efficient Cryptography — SEC 1: Recommended Elliptic Curve Domain Parameters. Version 1.0 (September 2000), http://www.secg.org/secg_docs.htm
7. Daly, A., Marnane, W., Kerins, T., Popovici, E.: An FPGA implementation of a GF(p) ALU for encryption processors. Elsevier - Microprocessors and Microsystems 28(5–6), 253–260 (2004)
8. de Dormale, G.M., Quisquater, J.-J.: High-speed hardware implementations of elliptic curve cryptography: A survey. J. Syst. Archit. 53(2-3), 72–84 (2007)
9. Diffie, W., Hellman, M.: New directions in cryptography. IEEE Trans. Inf. Theory 22, 644–654 (1976)
10. Eberle, H., Gura, N., Chang-Shantz, S.: A cryptographic processor for arbitrary elliptic curves over GF(2^m). In: Application-Specific Systems, Architectures, and Processors (ASAP), pp. 444–454 (2003)
11. ECRYPT. eBATS: ECRYPT Benchmarking of Asymmetric Systems. Technical report (March 2007), http://www.ecrypt.eu.org/ebats/
12. ElGamal, T.: A public key cryptosystem and a signature scheme based on discrete logarithms. IEEE Trans. Inf. Theory 31, 469–472 (1985)
13. Gaudry, P., Thomé, E.: The mp$\mathbb{F}q$ library and implementing curve-based key exchanges. SPEED: Software Performance Enhancement for Encryption and Decryption, 49–64 (2007)
14. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer, New York (2004)
15. Institute of Electrical and Electronics Engineers. IEEE P1363 Standard Specifications for Public Key Cryptography (2000)
16. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics—Doklady 7(7), 595–596 (1963)
17. Koblitz, N.: Elliptic curve cryptosystems. Mathematics of Computation 48, 203–209 (1987)
18. Lenstra, A.K., Verheul, E.R.: Selecting Cryptographic Key Sizes. Journal of Cryptology 14(4), 255–293 (2001)
19. McIvor, C., McLoone, M., McCanny, J.: An FPGA elliptic curve cryptographic accelerator over GF(p). In: Irish Signals and Systems Conference (ISSC), pp. 589–594 (2004)

20. Miller, V.: Uses of elliptic curves in cryptography. In: Williams, H.C. (ed.) CRYPTO 1985. LNCS, vol. 218, pp. 417–426. Springer, Heidelberg (1986)
21. National Institute of Standards and Technology (NIST). Recommended Elliptic Curves for Federal Government Use (July 1999), http://csrc.nist.gov/csrc/fedstandards.html
22. Orlando, G., Paar, C.: A High-Performance Reconfigurable Elliptic Curve Processor for GF(2^m). In: Paar, C., Koç, Ç.K. (eds.) CHES 2000. LNCS, vol. 1965, pp. 41–56. Springer, Heidelberg (2000)
23. Orlando, G., Paar, C.: A scalable GF(p) elliptic curve processor architecture for programmable hardware. In: Koç, Ç.K., Naccache, D., Paar, C. (eds.) CHES 2001. LNCS, vol. 2162, pp. 356–371. Springer, Heidelberg (2001)
24. Satoh, A., Takano, K.: A scalable dual-field elliptic curve cryptographic processor. IEEE Transactions Computers 52, 449–460 (2003)
25. Savas, E., Tenca, A.F., Ciftcibasi, M.E., Koc, C.K.: Multiplier architectures for GF(p) and GF(2^n). IEE Proc. Comput. Digit. Tech. 151(2), 147–160 (2004)
26. Solinas, J.A.: Generalized Mersenne Numbers. Technical report (September 09, 1999)
27. Suzuki, D.: How to maximize the potential of FPGA Resources for Modular Exponentiation. In: Paillier, P., Verbauwhede, I. (eds.) CHES 2007. LNCS, vol. 4727, pp. 272–288. Springer, Heidelberg (2007)
28. Weimerskirch, A., Paar, C.: Generalizations of the Karatsuba Algorithm for Efficient Implementations. Technical report, Ruhr-Universität-Bochum, Germany (2003)
29. Xilinx. Xilinx Virtex 4, 5 and Spartan 3A FPGAs (2008), http://www.xilinx.com/products/silicon_solutions/