A Parallel Architecture for Fast Computation of Elliptic Curve Scalar Multiplication over $GF(2^m)$

Nazar A. Saqib, Francisco Rodríguez-Henriquez and Arturo Díaz-Pérez
Computer Science Section, Electrical Engineering Department
Centro de Investigación y de Estudios Avanzados del IPN
Av. Instituto Politécnico Nacional No. 2508, México D.F.
nabbas@computacion.cs.cinvestav.mx
francisco,adiaz@cs.cinvestav.mx

Abstract

This paper presents a generic parallel architecture for fast elliptic curve scalar multiplication over binary extension fields. We show how the parallel strategy followed in this work leads to high performance designs. We also implemented the proposed architecture on reconfigurable hardware devices where the predicted expeditious performance figures were actually obtained. The results achieved show that our proposed design is able to compute $GF(2^{191})$ elliptic curve scalar multiplication operations in 56.44 μSecs.

1 Introduction

On October 24, 2003, the US National Security Agency (NSA) signed an agreement to purchase a license for using elliptic curve cryptography (ECC) systems as a standard means of securing all its classified documents. The agency plans to use a 512-bit key for the ECC system, which is the equivalent of an RSA key of 15,360 bits. This is the first time ever that the NSA has endorsed any sort of public-key cryptography system. [3]

Elliptic Curve Cryptosystems (ECC) were first proposed in 1985 independently by N. Koblitz [9] and V. Miller [12]. Since then it has been consistently proved that elliptic curves offer more security by key length than any other major public key cryptosystem.

Although elliptic curves can be defined over real numbers, complex numbers, and any other field, from the cryptographic point of view, we are only interested on elliptic curves defined over finite fields. Indeed, since a quite long time it has been known that when an non-singular elliptic curve is defined over a finite field, the points on the curve together with a *point addition* operation form an Abelian group [9].

The most important operation for elliptic curve cryptosystems is the so-called *Scalar multiplication* operation. Let n be a positive integer and P a point on an elliptic curve. Then the scalar multiple $Q = nP$ is the point resulting of adding $n-1$ copies of P to itself. The security of the cryptosystem relies on the discrete logarithm problem that can be defined as follows,

Given two points Q and P that belong to the curve, the problem to find a positive scalar such that the equation $Q = nP$ holds, is referred as the discrete logarithm problem in this group. Solving the discrete logarithm problem over elliptic curves is believed to be an extremely hard mathematical problem, much harder than its analogous one defined over finite fields of the same size.

Since ECC proposal in 1985, a vast amount of investigation has been carried out in order to obtain efficient ECC implementations [4]. Numerous fast implementations have been reported in all of them, software [11, 7, 21], VLSI [18, 19] and reconfigurable hardware [15, 20, 6, 5, 2] platforms with variable time and space performance results.

In order to compute elliptic curve scalar multiplication, the vast majority of the aforementioned designs use the three-layer hierarchical strategy depicted in Figure 1. This way, high performance implementations of elliptic curve cryptography depend heavily on the efficiency in the computation of the underlying finite field arithmetic operations included in the first layer of the model.

On the other hand and as it will be shown in this paper, each one of the three layers included in Figure 1 are suitable for being implemented using parallel or semi-parallel approaches. Although parallel architectures can potentially trade area in exchange of speed, to our knowledge, only in [20, 2] authors have explicitly essayed a parallel strategy to compute elliptic curve scalar multiplication.

In this paper we present a highly parallel architecture especially tailored to obtain an ultra fast implementation of

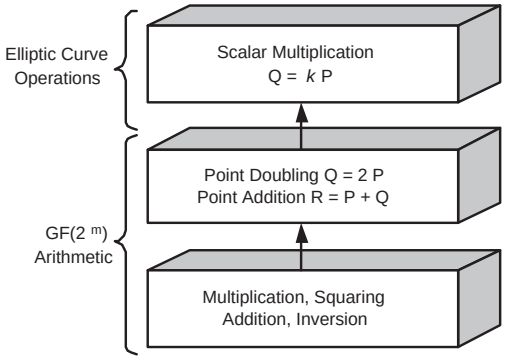

Figure 1. Three-Layer Model for Elliptic Curve Scalar Multiplication

the elliptic curves scalar multiplication operation. Special attention is devoted to the problem of finding efficient parallel architectures for the two upper layers of Figure 1. It will be shown that standard Xilinx VirtexE 3200 devices have enough resources to accommodate the realization of the parallel architecture proposed here. We describe all the implementation details of an architecture designed to compute $GF(2^{191})$ elliptic curve scalar multiplications in 56.44 μSecs. That constitutes a remarkable fast timing figure since to the best of our knowledge, before this work no other design has reported timings figures under 100 μSecs for computing elliptic curve point multiplications.

The rest of this paper is organized as follows. In section §2 we describe parallel strategies for the implementations of the second and third layers of Figure 1, i.e., point addition and point doubling operations based on the algorithm devised by [11]. Section §3 describes an architectural design for the implementation of elliptic curve scalar multiplication and parallel techniques for implementing finite field arithmetic in GF(2^{191}). FPGA's implementation of elliptic curve scalar multiplication and implementation results are also presented in this section. Performance comparison is made in section 4. Finally, in section §5 some conclusions remarks as well as future work are drawn. References are presented at the end.

2 Montgomery Point Multiplication

In this section we briefly discuss the algorithm used to compute point addition and point doubling and ultimately elliptic curve point multiplication. We follow the notation given in [11].

2.1 The Montgomery Algorithm

Let $P(x)$ be a degree-m polynomial, irreducible over $GF(2)$. Then $P(x)$ generates the finite field $F_q = GF(2^m)$ of characteristic two. A non-supersingular elliptic curve $E(F_q)$ is defined to be the set of points $(x, y) \in GF(2^m) \times GF(2^m)$ that satisfy the affine equation,

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

Where a and $b \in F_q, b \neq 0$, together with the point at infinity denoted by *0*.

Let $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ be two points that belong to the curve 1. Then $P+Q = (x_3, y_3)$ and $P-Q = (x_4, y_4)$, also belong to the curve and it can be shown that x_3 is given as [7],

$$x_3 = x_4 + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2}\right)^2; \tag{2}$$

Hence we only need the x coordinates of P, Q and $P-Q$ to exactly determine the value of the x-coordinate of the point $P + Q$. Let the x coordinate of P be represented by X/Z. Then, when the points $2P = (X_{2P}, Y_{2P}, Z_{2P})$ and $P + Q = (X_3, Y_3, Z_3)$ are converted to projective coordinate representation their coordinates can be computed as [11],

$$\begin{aligned} X_{2P} &= X^4 + b \cdot Z^4; \\ Z_{2P} &= X^2 \cdot Z^2; \\ Z_3 &= (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2; \\ X_3 &= x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1); \end{aligned} \tag{3}$$

Input: $k = (k_{n-1}, k_{n-2}....., k_1, k_0)_2$ with $k_{n-1} = 1$, $P(x, y) \in E(F_{2^m})$
Output: $Q = kP$

1. Set $X_1 \leftarrow x$, $Z_1 \leftarrow 1$, $X_2 \leftarrow x^4 + b$, $Z_2 \leftarrow x^2$
2. For i from $n-2$ downto 0 do
3. if ($k_i = 1$) then
4. Madd(X_1, Z_1, X_2, Z_2), Mdouble(X_2, Z_2)
5. else
6. Madd(X_2, Z_2, X_1, Z_1), Mdouble(X_1, Z_1)
7. Return($Q = M_{xy}(X_1, Z_1, X_2, Z_2)$)

Figure 2. Montgomery point multiplication

The algorithm shown in Figure 2 computes elliptic point multiplication over $GF(2^m)$ based on the formulae of equation 3 [7]. Its approximate running time is $6mM$ where M represents a field multiplication operation. In the next subsection we discuss how to obtain an efficient parallel implementation of the above algorithm.

2.2 Point addition and point doubling

Let $P(x, y) \in E(F_{2^m})$ be a point defined on the curve E, then the computation of point addition Madd(X_1, Z_1, X_2, Z_2) can be obtained from the execution of the sequence indicated in 4 that was directly obtained from 3.

$$\begin{array}{ll} T_2 & = x \\ P & = X_1 \times Z_2 \\ Q & = Z_1 \times X_2 \\ R & = P + Q \\ Z' & = R^2 \\ M & = P \times Q \\ N & = T_2 \times Z' \\ X' & = M + N \end{array} \quad (4)$$

Thus, the point addition computation consists on 4 multiplications, 2 additions and only one squaring.

The computational complexity of Point doubling (Mdouble) is simpler than the one of point addition. The following equation is the sequence of instructions needed to compute a single point doubling operation Mdouble(X,Y).

$$\begin{array}{ll} T_1 & = c \\ S & = X_2^2 \\ T & = Z_2^2 \\ Z_3 & = S \times T \\ W & = T \times T_1 \\ U & = W^2 \\ V & = S^2 \\ X_3 & = U + V \end{array} \quad (5)$$

The algorithm consists of only 2 multiplications, 4 squarings and one addition.

2.3 Montgomery point multiplication: A parallel approach

As it was mentioned earlier in the introduction section, parallel implementations of the three-layer architecture depicted in Figure 1 constitutes the main interest of this paper. We will briefly discuss how to do so in the case of the first layer in subsection §3.1. However, hardware resource limitations restrict us from attempting a fully parallel implementation of second and third layers. Thus, a compromising strategy must be adopted to exploit parallelism at second and third layers. Several options to do so are shown in Table 1.

Table 1 presents four different options that we can possibly follow in order to parallelize the algorithm of Figure 2. As is customary, the computational costs shown in table 1 are normalized with respect to the required number of field multiplication operations.

Table 1. $GF(2^m)$ Elliptic Curve Point Multiplication Computational Costs

Strategy		Required No. of Field Multipliers	EC Operation Cost		Total Number of Field Multiplications
2nd Layer	3rd Layer		Doubling	Addition	
S	S	1	$2M$	$4M$	6mM
S	P	2	$2M$	$4M$	4mM
P	S	2	M	$2M$	3mM
P	P	4	M	$2M$	2mM

Due to area restrictions we can afford to accommodate in our design, up to two fully parallel field multipliers. Hence, we have no option but to parallelize either the second or the third layer but not both, meaning that the first and fourth options are therefore out of the scope of this work. Both second and third options seem to be feasible; however, third option proves to be more attractive as it demonstrates better timing performance at the same area cost as compared to the second option. As it is indicated in the third row of Table 1, the estimated computational cost of our elliptic curve Point multiplication implementation will be of only $3m$ field multiplications.

In next section we discuss how this approach can be carried out on hardware platforms.

3 Architectural design

In this section, a generic parallel architecture for computing elliptic curve scalar multiplication on hardware platforms is described. The proposed architecture is based on a parallel-sequential approach of the Montgomery algorithm of Figure 2, discussed in the previous section. That approach corresponds to the one outlined in the third row of Table 1.

Figure 3 represents a generic architecture proposed for elliptic curve scalar multiplication by implementing equation 5 and equation 4 in just three clock cycles.

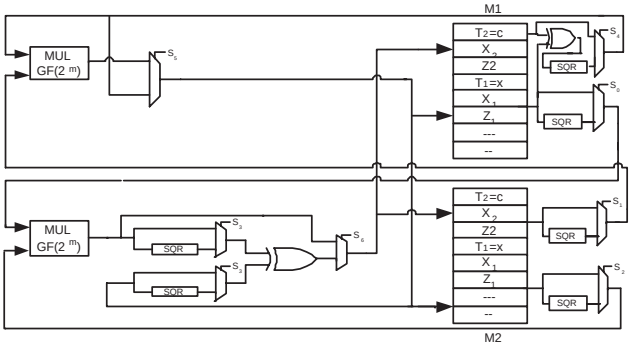

Figure 3. kP scalar multiplication

The parallel architecture shown in Figure 3 utilizes two multipliers in GF(2^m) together with two memory blocks to retrieve and store operands. A two-port RAM is the suggested configuration for the memory blocks. This configuration makes possible to have two independent data accesses simultaneously. Some combinational logic units comprising squarers, XOR operators, multiplexers are located before or after the two multipliers to perform pre or post computations in order to achieve the required final output.

Referring to the algorithm of Figure 2, each bit of vector k is tested from left to right (in descending order) where both of them, Madd and Mdouble operations are executed no matter if the test-bit is zero or one. The difference lies in the order of the passed arguments. if the test bit is '1' then Mdouble(X_2, Z_2) and Madd(X_1, Z_1, X_2, Z_2) are performed. Otherwise Mdouble(X_1, Z_1) and Madd(X_2, Z_2, X_1, Z_1) are computed. Madd and Mdouble blocks are computed according to Equations 5 and 4.

Table 2 demonstrates the algorithm flow to complete both point addition and point doubling operations in three normal clock cycles if test-bit is one. M1 and M2 are two memory blocks and each block has two input ports $PT1$ and $PT2$. The notation is the same used in Equations 5 and 4. It must be noted that the modular product computed by the multipliers is not always directly stored in the RAMs, as some logical operations required by the algorithm need to be performed. The same can be said with respect to the parameters extracted from RAMs into the multiplier's inputs.

Table 2. kP computation, if test-bit is '1'

Cycle	Read				Write	
	M1		M2		M1/M2	
	$PT1$	$PT2$	$PT1$	$PT2$	$PT1$	$PT2$
1	X_1	Z_2	Z_1	X_2	P	Q
2	X_2	Z_2	Z_2	T_1	Z_2=Z_3	X_2=X_3
3	P	Q	Q	T_2	X_1=X'	Z_1=Z'

If the test-bit is zero, then the order for the arguments passed is reversed but the storage locations remain the same. Table 3 details all the computational steps needed to execute the operation in three normal cycles. The same notation from Equations 5 and 4 was adopted.

The resultant vectors X_1,Z_1,X_2,Z_2, are therefore updated for the next cycle. As it takes three cycles to complete point addition and point doubling for one bit, $191 \times 3 \times T$ is the time taken for 191-bit test vector, where T is the allowed time period.

The generic architecture of Figure 3 can be extended for implementations of elliptic curve scalar multiplication on hardware platforms like FPGAs, VLSI, with minor modifications. In the rest of this section, it is explained how the proposed architecture can be optimized for FPGA devices to achieve high timing performances.

Table 3. kP computation, if test-bit is '0'

Cycle	Read				Write	
	M1		M2		M1/M2	
	$PT1$	$PT2$	$PT1$	$PT2$	$PT1$	$PT2$
1	X_2	Z_1	Z_2	X_1	P	Q
2	X_1	Z_1	Z_1	T_1	Z_1=Z_3	X_1=X_3
3	P	Q	Q	T_2	X_2=X'	Z_2=Z'

3.1 Finite Field Arithmetic

In this subsection we address the problem of how to implement efficiently finite field operations in reconfigurable hardware. In particular, we study how to implement two of the blocks shown in Figure 3: field multiplication and field squaring.

Let the field $GF(2^m)$ be constructed using the irreducible polynomial $P(x)$, of degree m, and let A, B be two elements in $GF(2^m)$ given in the polynomial basis as $A = \sum_{i=0}^{m-1} a_i x^i$ and $B = \sum_{i=0}^{m-1} b_i x^i$, respectively, with $a_i, b_i \in GF(2)$.

By definition, the field product $C' \in GF(2^m)$ of the elements $A, B \in GF(2^m)$ is given as

$$C'(x) = A(x)B(x) \bmod P(x). \quad (6)$$

However, in the approach followed in this work, equation (6) is computed in two steps: polynomial multiplication followed by modular reduction.

Let $A(x), B(x), C'(x) \in GF(2^m)$ and $P(x)$ be the irreducible polynomial generating $GF(2^m)$. In order to compute (6) we first obtain the product polynomial $C(x)$ of degree at most $2m-2$, as

$$C(x) = A(x)B(x) = (\sum_{i=0}^{m-1} a_i x^i)(\sum_{i=0}^{m-1} b_i x^i) \quad (7)$$

Then, in the second step, a reduction operation is performed in order to obtain the $m-1$ degree polynomial $C'(x)$, which is defined as

$$C'(x) = C(x) mod P(x). \quad (8)$$

In the rest of this subsection we show how to compute equation (7) efficiently, considering two separate cases. First, we describe an efficient method to compute polynomial squaring, which is a particular case of polynomial multiplication. Then, a practical reconfigurable hardware

implementation of Karatsuba-Ofman algorithm is briefly analyzed as one of the most efficient techniques to find the polynomial product of (7). Finally we discuss how to implement in a highly efficient way the reduction step of equation (8).

Squaring over $GF(2^m)$

Polynomial squaring over $GF(2)$ is a special case of polynomial multiplication, generally considered a costly operation in software. However in hardware platforms it is free of cost as it can be implemented occupying almost none hardware resource.

Let us assume that we have an element A given as $A = \sum_{i=0}^{m-1} a_i x^i$. Then the square of A is given as

$$\begin{aligned} C(x) &= A(x)A(x) &= A^2(x) \\ &&= (\sum_{i=0}^{m-1} a_i x^i)(\sum_{i=0}^{m-1} a_i x^i) \\ &&= \sum_{i=0}^{m-1} a_i x^{2i}. \end{aligned}$$

From the above equation, we immediately conclude that the first $k < m$ bits of A completely determine the first $2k$ bits of A^2. Notice also that half the bits of A^2 (the odd ones) are zeroes. This property happens to benefit hardware implementations as computation for polynomial squaring can be accomplished by just placing a zero value (connection to ground) at each alternative position of the original bits as shown in Figure 4. The implementation has a computational complexity O(1), hence its cost can be neglected.

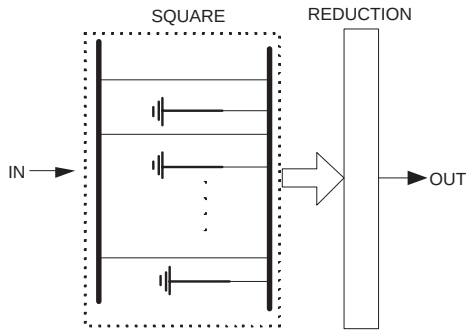

Figure 4. Squaring Circuit

Multiplication over $GF(2^m)$

Several architectures have been reported for multiplication in $GF(2^m)$. For example, efficient bit-parallel multipliers for both canonical and normal basis representation have been proposed in [8, 22, 13, 23, 2]. All these algorithms exhibit a space complexity $O(m^2)$. However, there are some asymptotically faster methods for finite field multiplications, such as the Karatsuba-Ofman algorithm [16]. Discovered in 1962, it was the first algorithm to accomplish polynomial multiplication in under $O(m^2)$ operations [1]. Karatsuba-Ofman multipliers may result in fewer bit operations at the expense of some design restrictions, particularly in the selection of the degree of the generating irreducible polynomial m.

In this research work we utilized a variation of the classic Karatsuba-Ofman Multiplier called *binary Karatsuba multipliers* that was first presented in [17]. Binary Karatsuba multipliers can be efficiently utilized regardless the form of the required degree m.

Let us consider the multiplication of two polynomials $A, B \in GF(2^m)$ with $m = rn = 2^k n$ (n is an integer), where A and B can be expressed as $A = x^{\frac{m}{2}} A^H + A^L$ and $B = x^{\frac{m}{2}} B^H + B^L$, then by using classical Karatsuba multiplier, the product of A and B can be expressed as:

$$\begin{aligned} C &= AB = A^H B^H x^m + \\ &+ (A^H B^H + A^L B^L + (A^H + A^L)(B^H + B^L)) x^{\frac{m}{2}} + \\ &+ A^L B^L \end{aligned} \tag{9}$$

Three multiplications each one of $2^{\frac{m}{2}}$ bits are therefore used to compute C.

Let us consider now the case (relevant for elliptic curve cryptography) where m is a prime, that can be expressed as $m = 2^k + d$, where d represents some *leftover* bits. One could be tempted to use direct Karatsuba algorithm by promoting m to 2^{k+1}. However this approach will clearly causes a wastage of extra arithmetic operations as all $2^k - d$ most significant bits are zeroes. Binary Karatsuba algorithm strategy suggests not to promote $2^k + d$ to 2^{k+1}, but instead perform multiplications separately for 2^k and d where only d is promoted to $2^{k'}$ where k' is an integer [17].

As a design example, consider the binary Karatsuba multiplier shown in Figure 5. That circuit computes the polynomial multiplication of the elements A and $B \in GF(2^{191})$. Notice that for this case $m = 191 = 2^k + d = 2^7 + 63$. We can do much better by assuming that the d = 63 most significant leftover bits are 64.

Once the polynomial multiplication/squaring over $GF(2^m)$ is completed, reduction must be performed as is explained in the remaining part of this subsection

Reduction

Let $A(x), B(x) \in GF(2^m)$ with irreducible polynomial $P(x)$ and we assume that the computation of polynomial

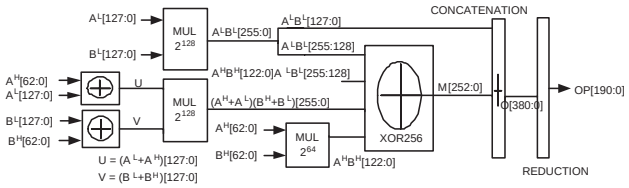

Figure 5. Karatsuba Multiplier

product $C(x)$ has already been made by using any of the two methods described in previous two subsections, then the modular product C' can be achieved by XOR operations only. Recall that the polynomial product C and the modular product C', have $2m-1$ and m, coordinates, respectively, i.e.,

$$\begin{aligned} C &= [c_{2m-2}, c_{2m-3}, \ldots, c_{m+1}, c_m, \ldots, c_1, c_0]; \\ C' &= [c'_{m-1}, c'_{m-2}, \ldots, c'_1, c'_0]. \end{aligned} \tag{10}$$

Figure 6 shows how to implement on reconfigurable hardware the reduction strategy for a general generating polynomial $P(x) = x^m + x^n + 1$. As it was mentioned before, the reduction step involves XOR and overlap operations. Notice that if we assume that $P(x)$ is a trinomial, then reduction becomes very economical as XOR operation fits well in 4-input/1-output typical structures of FPGA devices.

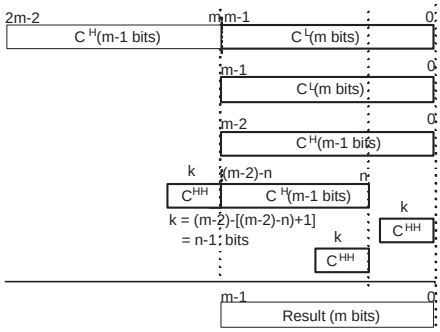

Figure 6. Reduction Diagram

Although the strategy shown in Figure 6 is completely general, for the purposes of this research work we utilized a fixed irreducible generating polynomial, namely, $P(x) = x^{191} + x^9 + 1$.

3.2 FPGA's implementation of Elliptic Curve Scalar Multiplication

For our FPGA implementation two 191-bit field multipliers were used in combination with eight multiplexers and six field squarer blocks. The design can be accommodated in a VirtexE XCV3200 Xilinx device. The VirtexE family of devices contains more than 280 fast access memory modules BlockRams (BRAMs). A dual port BRAM can be configured into two independent single port BRAMs. This special feature was exploited in our design in order to store and retrieve data from different memory locations irrespectively of the input port utilized.

Two similar memory blocks each of 12 BRAMs were used. It can be shown that only two BRAMs are sufficient, since the combination of 12 BRAMs actually allows storing and retrieving word lengths of 191 bits. The usage of those 12 BRAM helps to reduce the design complexity by saving a lot of register and multiplexer operations.

As is shown in Figure 3, Two pairs of multiplicands are read from the two BRAM blocks and multiplications are then computed in parallel by using the two available multipliers. Thus, six field multiplications (four for point addition and two for point doubling) can be completed in just three clock cycles. Since Read/Write operations for BRAMs are executed within one normal clock cycle, two separated clocks were designed. A master clock was used for the BRAMs, and then that clock was divided by two to serve as a second master clock for the rest of the design.

A control unit (CU) is designed to provide correct data paths as described in Table 2 and Table 3. It generates select signals (s_i) for the multiplexers as well as addresses to the BRAMs. The CU also testify each bit of the test vector k from left to right (in descending order) and then set correct data paths for reading and writing to the BRAMs before and after multiplication operation in a synchronized way. The test-bit is verified at the rising edge of 3rd clock cycle just after completing six multiplications for point addition and doubling, the data paths are ,therefore, set for the next clock cycle to start EC operations without loosing any cycle. The resultant vectors are up-dated at the completion of EC operations for each bit of test vector k . The loop is therefore executed 191 times to complete kP computations.

3.3 Implementation summary

All finite field arithmetic and then kP computational architectures were implemented on VirtexE XCV3200 by using Xilinx Foundation Tool F4.1i for design entry, synthesis, testing, implementation and verification of results. Table 4 represents timing performances and occupied resources by the said architectures.

Elliptic curve point addition and point doubling do not participate directly as a single computational unit in this design however parallel computations for both point addition and point doubling are designed together as it was shown in Figure 3. Both point addition and point doubling occupy 18300 (56.39 %) CLB slices and it takes $100.1\eta s$ (9.99

Table 4. Summary of Implemented designs

Design	Device (XCV)	CLB slices	Timings
Binary Karatsuba Multiplier $GF(2^{191})$	3200E	8721	$43.1\eta s$
Point addition + Point doubling $GF(2^{191})$(Fig 3)	3200E	18300	$300.3\eta s$
Point Multiplication $GF(2^{191})$	3200E 26 BRAMs	18314	$56.44\mu s$

MHz) for one computational cycle. As it is earlier mentioned that three cycles are used for computing both point addition and point doubling (six multiplications), $300.3\eta s$ is the total consumed time. Finally, point multiplication is performed in $56.44\mu s$ which is m (m=191 for our case) times the computational time for point addition and point doubling. It costs 18314 (56%) CLB slices and 24 (11%) BRAMs which act like registers for storing results and for saving numerous multiplexer operations. Our Binary Karatsuba Multiplier in $GF(2^{191})$ occupies 8721 (26.87%)CLB slices and one field multiplication is performed in $43.1\eta s$.

4 Comparison

Table 5 provides a quick comparison of the existing FPGA's implementations of elliptic curve scalar multiplication over $GF(2^m)$. Table 5 sum-up the last three years state of the art implementations, where most of the works featured have been published this same year. The design at [10] is a microcoded EC processor implemented on Annapolis Microsytems Wildstar board. For this design, EC multiplication is executed in $4.3ms$, $8.3ms$ and $11.1ms$ for GF(2^{113}), GF(2^{155}), and GF(2^{173}) respectively. An efficient VLSI EC processor at [18] supports EC scalar multiplication both in GF(p) and GF(2^n). Achieved results for a 160-bit EC scalar multiplication are $1.21ms$ and $0.19ms$ for GF(p) and GF(2^n) respectively. Another reconfigurable system on chip ECC implementation is reported on a special architecure AT94K40 from Atmel that integerates various components including an AVR 8-bit RISC microcontroller core, several peripherals and up to 36K Bytes SRAM within a single chip. That design execute EC operation in just $1.4ms$. All other designs [14, 20, 6, 2] implements EC scalar multiplication on a single chip FPGA. Among them our design is well compared with the fastest design at [2] showing an improvement of 79.26% in execution time. That high performance is achieved not only because of the usage of a relatively large FPGA target device, but also because of efficient field multipliers optimized for critical paths. We also investigated structural arrangements related to the computation of point addition and point doubling according to the block diagram shown in Figure 3.

The design presented in [18] can handle arbitrary fields and elliptic curves without changing its hardware configuration, while those parameters have been fixed in our implementation. However the design in [18] was implemented on a traditional ASIC chip, where the flexibility for design changes is quite limited or many times even inexistent. In our approach on the other hand, taking advantage of the reconfigurability feature of the platform selected, we preferred to optimize the performance of our design for a given field while the possibility to reconfigure the design for other parameters can still be instrumented.

Table 5. $GF(2^m)$ Elliptic Curve Point Multiplication Hardware Performance Comparison

Reference	Field	Platform	kP
[10]	$GF(2^{113})$ $GF(2^{155})$ $GF(2^{173})$	Annapolis Micro Systems Wildstar board	$4.3ms$ $8.3ms$ $11.1ms$
[18]	$GF(2^{160})$	0.13μ CMOS ASIC	$0.19ms$
[14]	$GF(2^{167})$	XCV400E	$0.21ms$
[20]	$GF(2^{191})$	XCV4000XL	$11.82ms$
[6]	$GF(2^{163})$	XCV2000E	$0.143ms$
[6]	$GF(2^{193})$	XCV2000E	$0.187ms$
[5]	$GF(2^{113})$	AT94K40	$1.4ms$
[2]	$GF(2^{191})$	XCV1000BG	$0.27ms$
This work	$GF(2^{191})$	XCV3200E	$0.056ms$

5 Conclusions

In this work, a parallel generic design strategy for elliptic curve point multiplication is presented which can work efficiently on hardware platforms, reconfigurables or not. The architecture is then optimized for $GF(2^{191})$ which results an ultra fast elliptic curve point multiplication with a performance time of just $56.44\mu s$ which is more than 3.4 times faster than any other work reported in the literature. The architecture targets Xilinx VirtexE XCV3200 FPGA device and occupies 18314 CLB slices.

Although our design has specifically targeted the field generated by the irreducible polynomial $P(x) = x^{191} + x^9 + 1$, all the machinery discussed in this paper has make no assumption about the specific field targeted and hence, can be easily adapted to accommodate other designs with different field sizes.

As a whole, the architecture presents a promising approach for elliptic curve scalar multiplication which provides a balance between space and time. In fact, the rapid

growth of FPGA's technology makes possible to accommodate parallel implementations of cryptographic algorithms like elliptic curve scalar multiplication.

Future work includes further improvements in the design performance by cutting the critical path in all three layers, the implementation of other design strategies and comparison between them.

References

[1] E. Bach and J. Shallit. *Algorithmic number theory, Volume I: efficient algorithms*. Kluwer Academic Publishers, Boston, MA, 1996.

[2] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, and J. von zur Gathen. Reconfigurable implementation of elliptic curve crypto algorithms. In *Proc. of The 9th Reconfigurable Architectures Workshop (RAW-02)*, Fort Lauderdale, Florida, U.S.A., April 2002.

[3] Certicom. Certicom news. "http://www.certicom.com/", october 2003.

[4] D. V. Chudnovsky and G. V. Chudnovsky. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. *Advances in Applied Math.*, 7:385–434, 1986.

[5] M. Ernst, M. Jung, and F. M. et. al. A reconfigurable system on chip implementation for elliptic curve cryptography over $GF(2^n)$. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2523:381–399, August 2003.

[6] N. Gura, S. Shantz, and H. E. et. al. An end-to-end systems approach to elliptic curve cryptography. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2523:349–365, August 2003.

[7] D. Hankerson, J. Lopez-Hernandez, and A. Menezes. Software implementation of elliptic curve cryptography over binary fields. *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, 1965:1–24, August 2000.

[8] M. A. Hasan, M. Z. Wang, and V. K. Bhargava. A modified Massey-Omura parallel multiplier for a class of finite fields. *IEEE Transactions on Computers*, 42(10):1278–1280, November 1993.

[9] N. Koblitz. Elliptic curve cryptosystems. *Mathematics of Computation*, 48(177):203–209, January 1987.

[10] P. H. Leong and I. K. Leung. A microcoded elliptic curve processor using fpga technology. *IEEE Transactions on VLSI Systems*, 10(5):550–559, October 2002.

[11] J. Lopez and R. Dahab. Fast multiplication on elliptic curves over $GF(2^m)$ without precomputation. *Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings*, 1717:316–327, August 1999.

[12] V. Miller. Uses of elliptic curves in cryptography. *In H. C. Williams (editor)* Advances in Cryptology — CRYPTO 85 Proceedings *Lecture Notes in Computer Science*, 218:417–426, January 1985.

[13] M. Morii, M. Kasahara, and D. L. Whiting. Efficient bit-serial multiplication and the discrete-time Wiener-Hopf equation over finite fields. *IEEE Transactions on Information Theory*, 35(6):1177–1183, 1989.

[14] G. Orlando and C. Paar. A high-performance reconfigurable elliptic curve processor for $GF(2^m)$. *Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings*, 1965:41–56, August 2000.

[15] G. Orlando and C. Paar. A scalable $GF(p)$ elliptic curve processor architecture for programmable hardware. *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, 2162:348–363, May 2001.

[16] C. Paar. A new architecture for a paralel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–861, July 1996.

[17] F. Rodríguez-Henríquez and Ç. K. Koç. On fully parallel karatsuba multipliers for $GF(2^m)$. In *International Conference on Computer Science and Technology (CST 2003)*, Cancun, Mexico, May 2003.

[18] A. Satoh and K. Takano. A scalable dual-field elliptic curve cryptographic processor. *IEEE Transactions on Computers*, 52(4):449–460, April 2003.

[19] R. Schroeppel, C. Beaver, R. Gonzales, R. Miller, and T. Draelos. A low-power design for an elliptic curve digital signature chip. *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA, August 13-15, 2002, Revised Papers*, 2523:366–380, August 2003.

[20] N. Smart. The hessian form of an elliptic curve. *Cryptographic Hardware and Embedded Systems - CHES 2001, Third International Workshop, Paris, France, May 14-16, 2001, Proceedings*, 2162:118–125, May 2001.

[21] N. Smart and E. Westwood. Point multiplication on ordinary elliptic curves over fields of characteristic three. *Applicable Algebra in Engineering, Communication and Computing*, 13:485–497, 2003.

[22] B. Sunar and Ç. K. Koç. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):522–527, May 1999.

[23] H. Wu and M. A. Hasan. Low complexity bit-parallel multipliers for a class of finite fields. *IEEE Transactions on Computers*, 47(8):883–887, August 1998.