# Contents

# List of Figures

# List of Tables

# Chapter I

# A Generic Coprocessor For Elliptic Curve Scalar Multiplication on Hardware

Nazar A. Saqib [I.1], Francisco Rodríguez-Henriquez [I.2], Arturo Díaz-Pérez [I.3]

In this chapter we present a generic parallel architecture for the computation of the scalar multiplication over binary fields for two elliptic curve forms: Hessian form and Weierstrass non-singular form. The architecture was designed as general as possible trying to make no assumptions about the specific hardware platform to be used by the designers. The idea of using parallel strategies was considered in every design stage and implemented as much as our hardware resources allowed us to do it so. The design results reported in this work allow us to compute GF($2^{191}$) elliptic curve scalar multiplication operations for the Hessian and the Weierstrass non-singular form in about 114.7$\mu$ Secs and 57$\mu$ Secs, respectively.

[I.1] 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

[I.2] Computer Science Section, Electrical Engineering Department, Centro de Investigación y de Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, México D.F., francisco@cs.cinvestav.mx

[I.3] Computer Science Section, Electrical Engineering Department, Centro de Investigación y de Estudios Avanzados del IPN, Av. Instituto Politécnico Nacional No. 2508, México D.F., adiaz@cs.cinvestav.mx

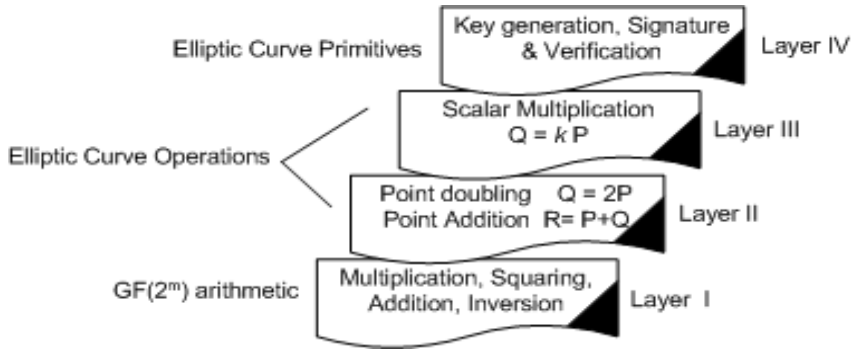


**Figure I.1.** Hierarchical Model for Elliptic Curve Cryptography

## I.1 Introduction

Over the past 17 years, many mathematical evidences have consistently shown that Elliptic Curve Cryptography (ECC) offers more security by key length than any other major public key cryptosystem. 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. Scalar multiplication is the main building block used in all the three fundamental ECC primitives: *Key Generation*, *Signature* and *Verification* schemes.

The security of elliptic curve systems is based on the intractability of the elliptic curve discrete logarithm problem (ECDLP) that can be formulated as follows. Given an elliptic curve $E$ defined over a finite field $F_q$ and two points $Q$ and $P$ that belong to the curve, where $P$ has order $r$, find a positive scalar $n \in [0, r-1]$ such that the equation $Q = nP$ holds. 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.

Several implementations have been reported so far [1, 2, 3, 4, 5, 6], and most of them utilize a four-layer hierarchical scheme such as the one depicted in Figure I.1. As a consequence, high performance implementations of elliptic curve cryptography directly depend on the efficiency in the computation of the three underlying layers of the model.

The main idea discussed throughout this chapter is that each one of the three bottom layers shown in Figure I.1 can be implemented using parallel strategies. Although parallel architectures offer an interesting potential for obtaining a high timing performance at the price of area, only in [4, 5] authors have explicitly attempted a parallel strategy to compute elliptic curve scalar multiplication.

The contribution of this chapter is the design of a generic parallel architecture especially tailored to obtain fast computation of the elliptic curves scalar multiplication operation. The architecture proposed here exploits the inherent parallelism of two elliptic curves forms defined over GF($2^m$): The Hessian form and the Weierstrass non-supersingular form.

The rest of this chapter is organized as follows. In Section I.2.2 we briefly describe the Hessian and Weierstrass Forms of an elliptic curve together with their corresponding group laws. We also include some comments about the different design options that one can take in order to obtain a parallel version of the point addition and doubling operators. In Section I.3 we describe the generic parallel architecture for scalar multiplication that constitutes the main contribution of this work. Then in Section §I.4 we give all the design details accomplished to implement the proposed architecture on an FPGA platform. Section I.5 includes a performance comparison of our design with other similar implementations previously reported. Finally, in Section I.6 some conclusions remarks as well as future work are drawn.

## I.2 Arithmetic Considerations

The bottom layer of Figure I.1 comprises the whole set of finite field arithmetic operations: multiplication, squaring, addition and inversion in GF($2^m$). The 2nd layer from bottom uses finite field arithmetic to construct two well-known EC arithmetic operations: point addition and point doubling. In this Section we briefly discuss some theoretical considerations of how to implement efficiently the operations required for our proposed design.

### I.2.1 Finite Field Arithmetic

By far the most important arithmetic field operation is field multiplication. In this Subsection, some optimizations techniques that allow efficient parallel multiplier structures are briefly discussed.

#### Multiplication over GF($2^m$)

Let $P(x)$ be the irreducible polynomial generating the Galois Field GF($2^m$) and $A(x)$, $B(x)$, $C(x) \in$ GF($2^m$). Then for the multiplication $C(x) = A(x)B(x)$ mod $P(x)$, we may first compute the product polynomial

$$C'(x) = A(x)B(x) = (\sum_{i=0}^{m-1} a_i x_i)(\sum_{i=0}^{m-1} b_i x_i) \tag{I.1}$$

and then reduce $C'(x)$ by using $P(x)$ to get the required result $C(x) \in$ GF($2^m$). In this work, we utilized a variation of the classic Karatsuba-Ofman Multiplier called *binary Karatsuba multipliers* [7]. Binary Karatsuba multipliers yield modular multiplier designs regardless the field size $m$ required. Let us consider the multiplication of two polynomials $A, B \in$ GF($2^m$), where $A$ and $B$ can be expressed as $A = x^{\frac{m}{2}} A^H + A^L$ and $B = x^{\frac{m}{2}} B^H + B^L$, two cases for $m$ might be considered.

Case 1 (Optimal case): $m = 2^k$ : then by using the classical Karatsuba-Ofman multiplier approach, the product of $A$ and $B$ can be expressed as:

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

Three polynomial multiplications each one of $\frac{m}{2} = 2^{k-1}$ bits are therefore used to compute C. By reusing recursively last equation one can obtain highly efficient multipliers blocks. Let MUL($2^k$) denote a m-bit Karatsuba-Ofman multiplier block with $m = 2^k$, $k$ an integer.

Case 2: $m = 2^k + d$ (d denotes leftover bits): 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 $k = \lfloor \log_2(m) \rfloor$ and where $d$ represents some possible *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 $m = 2^k + d$ 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 given as $k = \lceil \log_2(m) \rceil$ [7]. Figure I.2 depicts the block diagram for binary Karatsuba Multiplier for an arbitrary degree m. As shown in Figure I.2, it needs two MUL($2^k$)

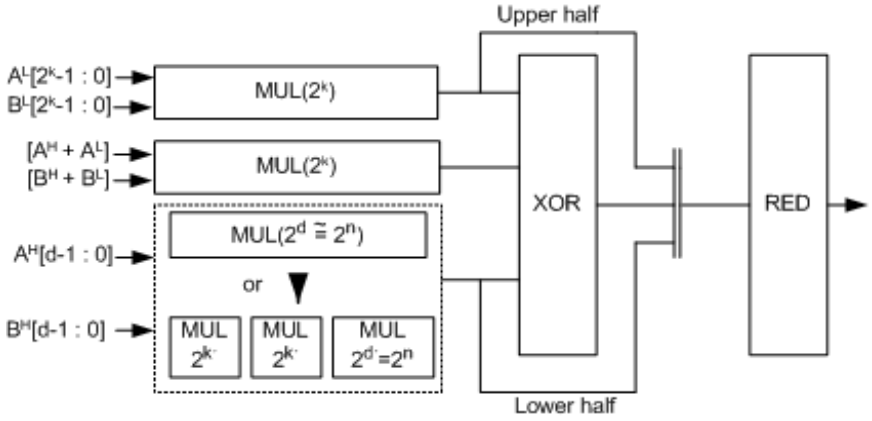


**Figure I.2.** Block diagram for binary Karatsuba Multiplier

multipliers plus one MUL($2^d$) that can be constructed with one MUL($2^n$) multiplier where n is the nearest power of 2 (e.g for d=63, n=6, $2^6 = 64$). As a design example, consider the binary Karatsuba multiplier shown in Figure I.3. That circuit computes the polynomial multiplication of the elements $A$ and $B \in GF(2^{191})$. Notice that for this case $m = 191 = 2^k + d = 2^7 + 63$. By assuming that the d $= 63$ most significant leftover bits are 64 we can build the required multiplier out of optimal Karatsuba-Ofman building blocks of the form MUL($2^n$). .

### Reduction

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

Let $A(x), B(x) \in GF(2^m)$ with irreducible polynomial $P(x)$ and we assume that the computation of polynomial product $C(x)$ has already been made by using

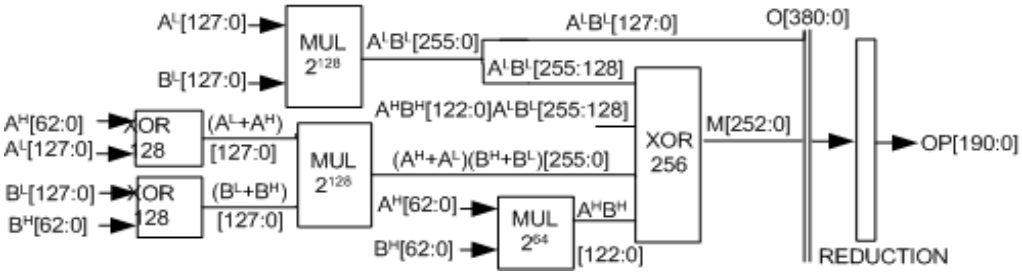


**Figure I.3.** Karatsuba Multiplier GF($2^{191}$)

the method described in the previous Subsection. Then the modular product $C'$ can be achieved by XOR operations only. Figure I.4 shows how to implement on reconfigurable hardware the reduction strategy for a general generating trinomial $P(x) = x^m + x^n + 1$, with $n < m/2$. 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. For the purposes of this research work we utilized a fixed irreducible generating trinomial, namely, $P(x) = x^{191} + x^9 + 1$.

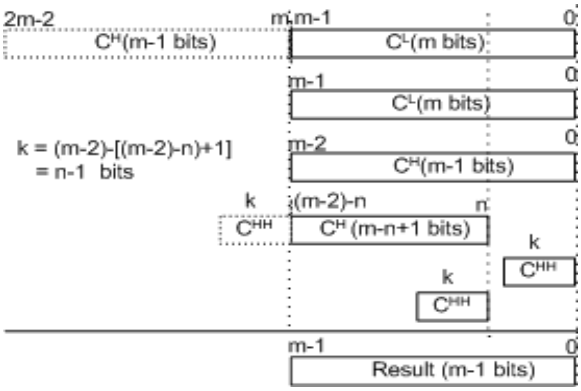


**Figure I.4.** Reduction Diagram

### I.2.2 ECC Arithmetic

In this Subsection we give the group law specifics of two elliptic curves forms defined over GF($2^m$): The form and the Weierstrass non-supersingular form. Then we describe the algorithms used to compute point addition and doubling for both forms. Finally, we include a discussion of how to compute the elliptic curve point multiplication operation by using parallel strategies.

### Hessian form

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 Hessian elliptic curve $E(F_q)$ is defined to be the set of points $(x, y, z) \in GF(2^m) \times GF(2^m)$ that satisfy the canonical homogeneous equation,

$$x^3 + y^3 + z^3 = Dxyz \tag{I.3}$$

Together with the point at infinity denoted by $\mathcal{O}$ and given by $(1, 0, -1)$.

The original form of the group law on curves in Hessian form belongs to Cauchy and was later simplified by Sylvester-Desboves [8].

Let $P = (x_1, y_1, z_1)$ and $Q = (x_2, y_2, z_2)$ be two points that belong to the plane cubic curve of Eq. I.3. Then we define $-P = (y_1, x_1, z_1)$ and $P + Q = (x_3, y_3, z_3)$ where,

$$\begin{array}{rcl} x_3 & = & {y_1}^2 x_2 z_2 - {y_2}^2 x_1 z_1 \\ y_3 & = & {x_1}^2 y_2 z_2 - {x_2}^2 y_1 z_1 \\ z_3 & = & {z_1}^2 y_2 x_2 - {z_2}^2 y_1 x_1 \end{array} \tag{I.4}$$

Provided that $P \neq Q$. The addition formulae of Eq. (I.4) might be parallelized using 12 field multiplications as follows [4],

$$\begin{array}{lll} \lambda_1 = y_1 x_2 & \lambda_2 = x_1 y_2 & \lambda_3 = x_1 z_2 \\ \lambda_4 = z_1 x_2 & \lambda_5 = z_1 y_2 & \lambda_6 = z_2 y_1 \\ s_1 = \lambda_1 \lambda_6 & s_2 = \lambda_2 \lambda_3 & s_3 = \lambda_5 \lambda_4 \\ t_1 = \lambda_2 \lambda_5 & t_2 = \lambda_1 \lambda_4 & t_3 = \lambda_6 \lambda_3 \\ x_3 = s_1 - t_1 & y_3 = s_2 - t_2 & z_3 = s_3 - t_3 \end{array} \tag{I.5}$$

Whereas the formulae for point doubling are giving by

$$\begin{array}{rcl} x_3 & = & y_1 \left( {z_1}^3 - {x_1}^3 \right); \\ y_3 & = & x_1 \left( {y_1}^3 - {z_1}^3 \right); \\ z_3 & = & z_1 \left( {x_1}^3 - {y_1}^3 \right). \end{array} \tag{I.6}$$

Where $2P = (x_3, y_3, z_3)$. The doubling formulae of Eq. (I.6) can be also parallelized requiring 6 field multiplications plus three field squarings for their computation. The resulting arrangement can be rewritten as [4],

$$\begin{array}{lll} \lambda_1 = {x_1}^2 & \lambda_2 = {y_1}^2 & \lambda_3 = {z_1}^2; \\ \lambda_4 = x_1 \lambda_1 & \lambda_5 = y_1 \lambda_2 & \lambda_6 = z_1 \lambda_3; \\ \lambda_7 = \lambda_5 - \lambda_6 & \lambda_8 = \lambda_6 - \lambda_4 & \lambda_9 = \lambda_4 - \lambda_5; \\ x_2 = y_1 \lambda_8 & y_2 = x_1 \lambda_7 & z_2 = z_1 \lambda_9; \end{array} \tag{I.7}$$

By implementing Eqs. (I.5) and (I.7), one can obtain the two building blocks needed for the implementation of the second layer shown in Figure I.1. Hence, provided that those two blocks are available, one can compute the third layer of Figure I.1 by using the well-known doubling and add algorithm of Figure I.5. That

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

1. Set $Q = P$
2. For i from $n - 2$ downto 0 do
3. $\quad Q = 2.Q$ (point doubling)
4. $\quad$ if $(k_i = 1)$ then
5. $\quad\quad Q = Q + P$ (point addition)
6. $\quad$ end if;
7. end for;

**Figure I.5.** Doubling & Add algorithm for Scalar Multiplication: MSB-First

sequential algorithm needs an average of $\frac{m}{2}$ point additions plus $m$ point doublings in order to complete one scalar multiplication computation.

Alternatively, we can use the algorithm of Figure I.6 that can potentially be implemented in parallel since in this case the point addition and doubling operations do not show any dependencies between them. Therefore, if we assume that the algorithm of Figure I.6 is implemented in parallel, one needs an average of just $\frac{m}{2}$ point additions plus $\frac{m}{2}$ point doublings in order to complete one scalar multiplication computation.

In Subsection I.2.3 we discuss how to obtain an efficient parallel-sequential implementation of the second and third layers of the model of Figure I.1.

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

1. Set $Q = 1$; $R = P$
2. For i from 0 to $n - 1$ do
3. $\quad$ if $(k_i = 1)$ then
4. $\quad\quad Q = Q + R$ (point addition)
5. $\quad R = 2.R$ (point doubling)
6. $\quad$ end if;
7. end for;

**Figure I.6.** Doubling & Add algorithm for Scalar Multiplication: LSB-First

### Weierstrass Non-Singular form and Montgomery Point Multiplication Algorithm

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

$$y^2 + xy \;=\; x^3 + ax^2 + b, \tag{I.8}$$

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 Eq. I.8. 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 [1],

$$x_3 = x_4 + \frac{x_1}{x_1 + x_2} + \left(\frac{x_1}{x_1 + x_2}\right)^2; \tag{I.9}$$

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 [9],

$$\begin{array}{rcl} X_{2P} & = & X^4 + b \cdot Z^4; \\ Z_{2P} & = & X^2 \cdot Z^2; \\ Z_3 & = & (X_1 \cdot Z_2 + X_2 \cdot Z_1)^2; \\ X_3 & = & x \cdot Z_3 + (X_1 \cdot Z_2) \cdot (X_2 \cdot Z_1); \end{array} \tag{I.10}$$

Then it can be shown that the computation of point addition as defined in Eq. I.10 can be obtained by performing a total of 4 field multiplications, 2 field additions and one field squaring. Similarly, the point doubling operation can be computed by performing only 2 field multiplications, 4 squaring operations and one field addition.

The algorithm of Figure I.7 computes the point multiplication operation using the group operation defined by Eq. I.10. The approximate running time of that algorithm in its sequential version is $6m$M where M represents a field multiplication operation. But just as we proceeded for the case of the Hessian form, Eq. I.10 can be computed in a highly parallel way. Assuming that we can perform up to two multiplications in parallel, the Montgomery group operation for point addition defined by Equation I.10 can be carried out as,

$$\begin{array}{lll} T_2 = x; & P = X_1 \times Z_2; & Q = Z_1 \times X_2; \\ R = P + Q; & Z' = R^2; & \\ M = P \times Q; & N = T_2 \times Z'; & X' = M + N; \end{array} \tag{I.11}$$

Requiring only two multiplication cycles (other operations are negligible). Under the same assumptions, the point doubling operation on the Montgomery formulae of Eq. I.10 can be obtained in virtually one clock cycle,

$$\begin{array}{lll} T_1 = c & S = X_2^2 & T = Z_2^2 \\ Z_3 = S \times T & W = T \times T_1 & U = W^2 \\ V = S^2 & X_3 = U + V & \end{array} \tag{I.12}$$

In the next Subsection we discuss different available options for the parallelization of the algorithm shown in figure I.7.

**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 I.7.** Montgomery point multiplication

### I.2.3 Parallel Strategies for Scalar Point Multiplication

As it was mentioned in the introduction Section, parallel implementations of the three underlying layers depicted in Figure I.1 constitutes the main interest of this chapter. We briefly described how to do so in the case of the first layer in Subsection §I.2.1. 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 I.1.

2

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

| Strategy | | Req. No. | EC Operation Cost | | T. N0. of | EC Operation Cost | | T. No. of |
|---|---|---|---|---|---|---|---|---|
| 2nd | 3rd | of Field | Hessian form | | Field | Montgomery Algorithm | | Field |
| Layer | Layer | Mults. | Doubling | Addition | Mults. | Doubling | Addition | Mults. |
| S | S | 1 | $6M$ | $12M$ | $18mM$ | $2M$ | $4M$ | $6mM$ |
| S | P | 2 | $6M$ | $12M$ | $12mM$ | $2M$ | $4M$ | $4mM$ |
| P | S | 2 | $3M$ | $6M$ | $9mM$ | $1M$ | $2M$ | $3mM$ |
| P | P | 4 | $6M$ | $6M$ | $6mM$ | $M$ | $2M$ | $2mM$ |

Table I.1 presents four of the many options that we can follow in order to parallelize the computation of scalar point multiplication. The computational costs

shown in Table I.1 are normalized with respect to the required number of field multiplication operations (since the computation time of squaring operations can be neglected in arithmetic over GF($2^m$)).

Due to area restrictions we can afford to accommodate up to two fully parallel field multipliers in our design. Thus, we can afford both, second and third options of Table I.1. However, third option is definitely more attractive as it demonstrates better timing performance at the same area cost. Therefore, and as it is indicated in the third row of Table I.1, the estimated computational cost of our elliptic curve Point multiplication implementation will be of $9m$ field multiplications in Hessian form. It costs only $3m$ field multiplications using the Montgomery algorithm for the Weierstrass form.

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

## I.3 Generic architecture for scalar point multiplication

Figure I.8 shows a generic structure for parallel implementation of elliptic curve scalar multiplication on hardware platforms. That structure is able to implement the parallel-sequential approach listed in the third row of Table I.1, assuming the availability of two GF($2^m$) multiplier blocks.

As shown in Figure I.8, the basic organization for the computation of elliptic curve scalar multiplication is comprised of four classes of blocks: GF($2^m$) multipliers, Combinational logic blocks and/or finite field arithmetic (i.e. squaring, etc.), Blocks for intermediate results storage and selection (i.e. registers, multiplexers, etc.), and a Control unit (CU). The CU is the default part of every hardware circuit to control the flow of data between different stages of the design. Registers are provided to save the intermediate results. The results are further multiplexed to provide correct operands to the multipliers for next stages. The CL blocks perform pre or post computations at the multipliers to obtain final results within pre-defined clock cycles.

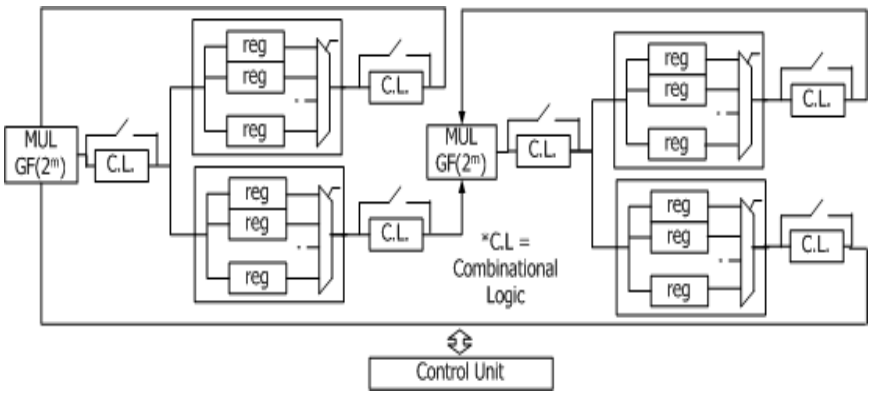


**Figure I.8.** Basic organization of elliptic curve scalar implementation

Figure I.8 presents a generic architecture that can be used for elliptic curve

scalar multiplication with minor modifications on different hardware platforms like VLSI and FPGAs. Although the selection of the components is totally platform dependent, the selection of some special devices might be helpful to reduce design complexity. For example, using read/write memories (RAMs) instead of several registers and multiplexers might provide more design modularity and efficiency. On the other hand, one cannot ignore the fact that the most costly operation in $kP$ computations by far is finite field multiplication over GF($2^m$). Therefore, efficient GF($2^m$) field multiplier blocks become indispensable in order to obtain fast and low-area implementations of elliptic curve scalar multiplication.

## I.4 Implementing Scalar Multiplication on Reconfigurable Hardware

Figure I.9 proposes a parallel structure for implementing the point multiplication algorithm discussed in Section I.2.2. It is a generic FPGA architecture based on the parallel-sequential approach for $kP$ computations discussed before. To implement

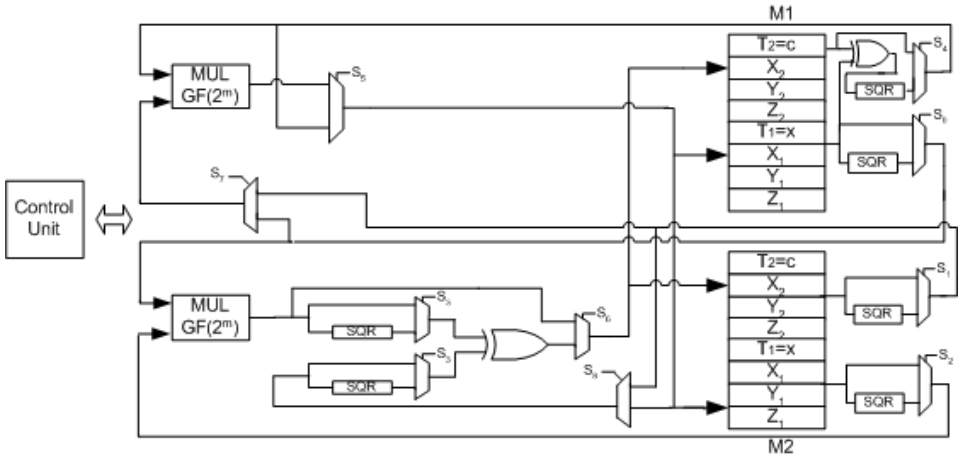


**Figure I.9.** Implementation of Scalar Multiplication on FPGA platforms

the memory blocks of Figure I.8, fast access FPGA's read/write memories Block-RAMs (BRAMs) are used. BRAMs are built-in memory blocks available in modern FPGAs. A dual port BRAM can be configured as a two single port BRAMs with independent data access. This special feature allows the saving of a considerable number of multiplexer operations as the required data is accessible independently from any of the two available input ports. Hence, two similar BRAMs blocks (each one of 12 BRAMs) provide four operands to the two multiplier blocks simultaneously. Since each BRAM contains 4k memory cells, two BRAM blocks are sufficient. The combination of 12 BRAMs however provide an access to 191-bit bus length. All control signals (read/write, address signals to the BRAMs and multiplexer enable signals) are generated by the control unit (CU). A master clock is directly fed to the BRAM block which is afterwards divided by two, serving as a master clock for the rest of the circuitry. The external multiplexers do apply pre and post computations

(squaring, XOR, etc.) on the inputs of the multipliers when they are required. In the next Subsection we discuss how the architecture of Figure I.8 can be used to compute Hessian scalar multiplication.

### I.4.1 Scalar Multiplication in Hessian form

According to Eq. (I.5) of Section I.2.2 we know that addition of two points in Hessian form consists of 12 multiplications, 3 squarings and 3 addition operations. Implementing squaring over GF($2^m$) is a trivial operation, so we can neglect it. Using the parallel architecture proposed in Figure I.9, point addition can be performed in 6 clock cycles using two GF($2^{191}$) multiplier blocks. For computing Hessian point addition according to Eq. (I.5), the sequence of the field multiplications must be followed as shown in Table I.2. In the architecture of Figure I.9, $M1$ and $M2$ are two memory (BRAMs) blocks, each one comprising of two independent ports $PT1$ and $PT2$. It should be noticed that the inputs/outputs of the multipliers are different from those read/write values at the memory blocks. It is due to pre or post computations required during the next clock cycle. Table I.2 lists values for the multiplications during read cycle and after the multiplications during write cycle.

**Table I.2.** Point addition in hessian form

| Cycle | Read | | | | Write | |
|---|---|---|---|---|---|---|
| | M1 | | M2 | | M1/M2 | |
| | $PT1$ | $PT2$ | $PT1$ | $PT2$ | $PT1$ | $PT2$ |
| 1 | $Y_1$ | $X_1$ | $X_2$ | $Y_2$ | $\lambda_1$ | $\lambda_2$ |
| 2 | $X_1$ | $Z_1$ | $Z_2$ | $X_2$ | $\lambda_3$ | $\lambda_4$ |
| 3 | $Z_1$ | $Z_2$ | $Y_2$ | $Y_1$ | $\lambda_5$ | $\lambda_6$ |
| 4 | $\lambda_1$ | $\lambda_2$ | $\lambda_6$ | $\lambda_5$ | $x_3$ | – |
| 5 | $\lambda_2$ | $\lambda_1$ | $\lambda_3$ | $\lambda_4$ | $y_3$ | – |
| 6 | $\lambda_5$ | $\lambda_6$ | $\lambda_4$ | $\lambda_3$ | $z_3$ | – |

Similarly Hessian point doubling implementation of Eq. (I.7) consists of 6 multiplications, 3 squarings and 3 additions. Table I.3 describes the algorithm flow implemented using the same architecture ( Figure I.9).

**Table I.3.** Point doubling in hessian form

| Cycle | Read | | | | Write | |
|---|---|---|---|---|---|---|
| | M1 | | M2 | | M1/M2 | |
| | $PT1$ | $PT2$ | $PT1$ | $PT2$ | $PT1$ | $PT2$ |
| 1 | $X_1$ | $Y_1$ | $X_1$ | $Y_1$ | $\lambda_4$ | $\lambda_9$ |
| 2 | $\lambda_9$ | $\lambda_4$ | $Z_1$ | $Z_1$ | $z_2$ | $\lambda_8$ |
| 3 | $\lambda 8$ | $\lambda_9$ | $Y_1$ | $X_1$ | $x_2$ | $y_2$ |

Let $m$ represents the number of bits and $M$ denotes a single finite field multiplication. Then the number of multiplications for one point addition and point

doubling are $6M$ and $3M$ respectively. Referring to the algorithm in Figure I.5, average of $(\frac{m}{2})6M$ and $3mM$ multiplications are needed for computing all $m$ bits of the vector $k$. Thus, 6mM are the total multiplication operations required for computing $kP$ scalar multiplication. In our case, m = 191 bits, the total number of field multiplications required by the algorithm are 1146. Let $T$ be the allowed clock period then $1146 \times T$ is the total time to complete elliptic curve scalar multiplication.

### I.4.2 Montgomery Point Multiplication

Referring to the algorithm of Figure I.7, each bit of vector $k$ is tested from left to right (in descending order). For each test bit (zero or one), both point addition (Madd) and point doubling (Mdouble) operations are performed. However, order of the arguments is reversed: if the test bit is '1', Mdouble$(X_2, Z_2)$, Madd$(X_1, Z_1, X_2, Z_2)$ are computed and Mdouble$(X_1, Z_1)$, Madd$(X_2, Z_2, X_1, Z_1)$ otherwise. Eqs. I.11 and I.12 describe algorithm flow for Madd and Mdouble operations respectively.

Tables I.4 and I.5 describe the multiplications performed for both point addition and point doubling operations in three normal clock cycles when test bit is '1' or '0' respectively. The notations used in Eqs. I.11 and I.12 for point addition and point doubling were kept the same. M1 and M2 represent two memory blocks (BRAMs) each one with two independent ports $PT1$ and $PT2$. Some required arithmetic operations (squaring etc.) need to be performed during read/write cycles at the memories before and after the multiplication operations.

**Table I.4.** $kP$ 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 | $T_2$ | $X_1$=$X'$ | $Z_1$=$Z'$ |

The resultant vectors $X_1$,$Z_1$,$X_2$,$Z_2$, are updated at the memories after the completion of point addition and doubling operations using 3 clock cycles for each bit. Total time for whole 191-bit test vector is therefore $191 \times 3 \times T$, $T$ represents allowed frequency.

**Table I.5.** $kP$ 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 | $T_2$ | $X_2$=$X'$ | $Z_2$=$Z'$ |

**Table I.6.** Design Implementation Summary

| Design | Device (XCV) | CLB slices | Timings |
|---|---|---|---|
| Binary Karatsuba Multiplier | 3200E | 8721 | $43.1\eta s$ |
| 1 Field Multiplication | | | $100.1\eta s$ |
| Point addition + Point doubling in Hessain Form | 3200E | 18300 | $300.3\eta s$ (if bit = '0')<br>$900.9\eta s$ (if bit = '1') |
| Point Multiplication in Hessian form | 3200E | 18314 & 24 BRAMs | $114.71\mu s$ |
| Point addition + Point doubling (Montgomery Point Multiplication) | 3200E | 18300 | $300.3\eta s$ (3 Multiplications) |
| Point Multiplication (Montgomery Point Multiplication) | 3200E | 18314 & 24 BRAMs | $57\mu s$ |

### I.4.3 Implementation Summary

All finite field arithmetic blocks and then the $kP$ computational architecture were implemented on a VirtexE XCV3200e-8bg560 device by using Xilinx Foundation Tool F4.1i for design entry, synthesis, testing, implementation and verification of results. Table I.6 lists 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 I.5. Both point addition and point doubling occupy 18300 (56.39 %) CLB slices and it takes $100.1\eta s$ (at a clock speed of 9.99 MHz) to complete one execution cycle. As it was mentioned, six and three cycles are needed for computing point addition and point doubling in Hessian form, respectively. Thus the total consumed time for computing each iteration of the algorithm of Figure I.5 is $900.9\eta$ if the corresponding bit is one and $300.3\eta s$ otherwise. Therefore, scalar point multiplication in Hessian form is the time to complete m/2 point additions (in average) and m point doublings. For our case m=191, the total time is therefore $(191/2)(600.6) + 191(300.3) = 114.71\mu s$. Similarly, two and one multiplications are needed to perform Montgomery point multiplication, thus consuming $100.1\eta s$ and $200.2\eta s$ for point doubling and point addition respectively. Each iteration of the algorithm thus consumes $300.3\eta s$ for 3 multiplications. For our case $m = 191$, total time is therefore $191(300.3) = 57\mu s$ for elliptic curve scalar multiplication. The architecture for elliptic curve scalar multiplication in both cases occupies 18314 (56%) CLB slices, 24 (11%) BRAMs and performs at the rate of $100.1\eta s$ (9.99 MHz). The design for Karatsuba Multiplier in GF($2^{191}$) occupies 8721 (26.87%) CLB slices and one field multiplication is performed in $43.1\eta s$.

## I.5 Performance comparison

Table I.7 provides a list of reported designs for elliptic curve scalar multiplication over GF($2^m$) implemented on FPGAs or VLSI. From that list, our designs obtained

the fastest computation time needing only 114.71$\mu s$ in Hessian form and 57$\mu s$ using Montgomery point multiplication algorithm.

**Table I.7.** GF($2^m$) Elliptic Curve Point Multiplication Hardware Performance Comparison

| Reference | Field | Platform | $kP$ ($\mu$Secs) |
|---|---|---|---|
| [6] | GF($2^{160}$) | 0.13$\mu$ CMOS ASIC | 190 |
| [10] | GF($2^{167}$) | XCV400E | 210 |
| [4] | GF($2^{191}$) | XCV4000XL | 11820 |
| [11] | GF($2^{163}$) | XCV2000E | 143 |
| [11] | GF($2^{193}$) | XCV2000E | 187 |
| [12] | GF($2^{113}$) | AT94K40 | 1400 |
| [5] | GF($2^{191}$) | XCV1000BG | 270 |
| EC scalar multiplication in Hessian Form | GF($2^{191}$) | XCV3200E | 114.71 |
| EC scalar multiplication (Montgomery Point Multiplication) | GF($2^{191}$) | XCV3200E | 57 |

## I.6 Conclusions

In this work, a generic architecture for elliptic curve point multiplication was presented. The proposed architecture is based on parallel implementations of each stage of the $kP$ computation process. Although the architecture was optimized for reconfigurable devices, it can be applied to other hardware platforms (such as VLSI) with minor modifications.

The structure presented in this chapter constitutes a generic architecture for the scalar multiplication in Hessian form as well as for Montgomery point multiplication defined over GF($2^{191}$). The resulting performance time for the scalar point multiplication operation in Hessian form is of 114.71$\mu s$, while it takes just 57$\mu s$ to complete Montgomery point multiplication. Therefore the Weierstrass form utilizing the Montgomery group formulation can be computed in about half the execution time consumed by the time needed in the case of the Hessian form.

Two major factors contribute in achieving high performances for our architecture. First, parallel strategies that were applied throughout the design ranging from finite field arithmetic to scalar point multiplication. The efficient field units (multipliers, squarers, etc) were designed and optimized for shortest possible data critical paths. Second, we investigated for the best use of those basic building blocks that includes the structural arrangements related to the computation of point addition and point doubling as described in Table I.1. As a whole, implementation of elliptic curve scalar multiplications on FPGA devices yields efficient architectures, showing a good balance between speed and time.

Future work includes the search for faster algorithms for elliptic curve scalar multiplication, the implementation of other design strategies and comparison between them.

# Appendix: Copyright

You must include your filled-in and signed copyright release form when your paper is accepted for publication. We must have this form before your paper can be published in the book. The copyright form is available either as a postscript file, "copyright.ps", as a PDF version, "copyright.pdf". The signed copyright should be sent to the editor-in-chief of the series by Fax to this number: +55 21 2587 7374.

# Index

# Author Index

# Bibliography


[1] D. Hankerson, J. Lopez-Hernandez, 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 (2000) 1–24.

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

[3] G. Orlando, 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 (2001) 348–363.

[4] 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 (2001) 118–125.

[5] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, 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., 2002.

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

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

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

[9] J. Lopez, 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 (1999) 316–327.

[10] G. Orlando, 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 (2000) 41–56.

[11] N. Gura, S. Shantz, 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 (2003) 349–365.

[12] M. Ernst, M. Jung, F. Madlener, 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
(2003) 381–399.