High-performance Elliptic Curve Cryptography by Using the CIOS Method for Modular Multiplication

Amine Mrabet, Nadia El-Mrabet, Ronan Lashermes, Jean-Baptiste Rigaud, Belgacem Bouallegue, Sihem Mesnager, Mohsen Machhout

High-performance Elliptic Curve Cryptography by Using the CIOS Method for Modular Multiplication

Amine MRABET[1,3,5], Nadia EL-MRABET[2], Ronan LASHERMES[7], Jean-Baptiste RIGAUD[2], Belgacem BOUALLEGUE[6], Sihem MESNAGER[1,4] and Mohsen MACHHOUT[3]

[1]University of Paris XIII, CNRS, UMR 7539 LAGA - France
[2]Ecole des Mines de St-Etienne, SAS-CMP - France
[3]University of Monastir $E\mu E$ Lab - Tunisia
[4] Télécom ParisTech
[5]National Engineering School of Tunis - Tunisia
[6]King Khalid University - Saudi Arabia
[7]LHS-PEC TAMIS INRIA-Rennes - France

Abstract Elliptic Curve Cryptography (ECC) is becoming unavoidable, and should be used for public key protocols. It has gained increasing acceptance in practice due to the significantly smaller bit size of the operands compared to RSA for the same security level. Most protocols based on ECC imply the computation of a scalar multiplication. ECC can be performed in affine, projective, Jacobian or others models of coordinates. The arithmetic in a finite field constitutes the core of ECC Public Key Cryptography. This paper discusses an efficient hardware implementation of scalar multiplication in Jacobian coordinates by using the Coarsely Integrated Operand Scanning method (CIOS) of Montgomery Modular Multiplication (MMM) combined with an effective systolic architecture designed with a two-dimensional array of Processing Elements (PE). As far as we know this is the first implementation of such a design for large prime fields. The proposed architectures are designed for Field Programmable Gate Array (FPGA) platforms. The objective is to reduce the number of clock cycles of the modular multiplication, which implies a good performance for ECC. The presented implementation results focuses on various security levels useful for cryptography. This architecture have been designed in order to use the flexible DSP48 on Xilinx FPGAs. Our architecture for MMM is scalable and depends only on the number and size of words.

Keywords: Hardware Implementation, ECC, Modular Multiplication, Montgomery Algorithm, CIOS method, Systolic Architecture, DSP48, FPGA

1 Introduction

The search for the most optimised architecture for arithmetic has always fascinated the embedded system world. In recent years this has been especially the case in finite fields for cyber security due to the invention of asymmetric encryption systems based on modular arithmetic operations. Throughout the history of cryptography for embedded systems, there has been a need for efficient architectures for these operations. The implementations must be cost-effective, both in terms of area and latency. Finite field arithmetic is the most important primitive of ECC, pairing and RSA. Since 1976, many

Public Key Cryptosystems (PKC) have been proposed and all these cryptosystems base their security on the difficulty of some mathematical problem. The hardness of this underlying mathematical problem is essential for security. Elliptic Curve Cryptosystems which were proposed by Koblitz [9] and Miller [15], RSA [21] and the Pairing-Based Cryptography[8] are examples of PKCs. All these systems require an efficient finite field multiplication. As a consequence, the development of efficient architecture for modular multiplication has been a very popular subject of research. In 1985, Montgomery has presented a new method for modular multiplication [17]. It's one of the most suitable algorithm for performing modular multiplications in hardware and software implementations. The efficient implementation of the Montgomery Modular Multiplication (MMM) in hardware was considered by many authors [3,6,7,18,19,23]. There is a variety of ways to perform the MMM, considering if multiplication and reduction are separated or integrated. A systolic array architecture [12,24] is one possibility for the implementation of the Montgomery algorithm in hardware, with a design both parallel and pipelined [3,18,19,20,23]. A similar work [20] has been done for binary fields (field characteristic is a power of 2) without having to deal with carry propagation as a consequence. These architectures use a Processing Elements (PE) array where each Processing Element performs arithmetic additions and multiplications. In accordance with the number of words used, the architecture can employ a variable number of PEs. The systolic architecture uses very simple Processing Elements (as in a pipeline). As a consequence, the systolic architecture decreases the needs for logic elements in hardware implementations. Our contribution is to combine a systolic architecture, which is assumed to be the best choice for hardware implementations, with the CIOS method of Montgomery modular multiplication. We optimize the number of clock cycles required to compute a n-bit MMM and we reduce the utilization of FPGA resources. We have implemented the modular multiplication in a fixed number of clock cycles. To the best of our knowledge, this is the first time that a hardware or a software multiplier of modular Montgomery multiplication, suitable for various security level, is performed in just 33 clock cycles. Furthermore, as far as we know, our work is the first one dealing with systolic architecture and CIOS method over large prime characteristic finite fields. Using our efficient MMM hardware implementation, we propose an efficient design for ECC operations: point addition and doubling. This paper is organized as follows: Section 2 discusses related state-of-the-art works. Section 3 presents the CIOS method of Montgomery Modular Multiplication algorithm. The proposed architectures and results are presented in Section 4 and Section 5. Finally, the conclusion is drawn in Section 6.

2 State of the art

2.1 Elliptic Curve Cryptography

The use of elliptic curves in cryptography has been independently introduced by Victor S. Miller [16] and N. Koblitz [10] during the 80s. The main advantage of ECC is that the bit sizes of the key are reduced for the same security level when comparing with a classical RSA algorithm [22]. For instance, at the AES 128-bit security level the key for ECC is

256 bits, while RSA requires a 3072-bit key. The main operation in ECC is the scalar multiplication over the elliptic curve. This scalar multiplication consists in computing $\alpha \times P$, for α an integer and P a point of an elliptic curve. When such an operation is implemented on an embedded system such as a FPGA, it is subject to constraints of area and speed. Efficient scalar multiplication arithmetic is hence a central issue for cryptography. The interested reader is referred to [2] for a good overview of the question.

ECC preliminaries Let E be an elliptic curve defined over $\mathbb{F}_p$ with $p > 3$ according to the following short Weierstrass equation:

$$E : y^2 = x^3 + ax + b \tag{1}$$

where $a, b \in \mathbb{F}_p$ such that $4a^3 + 27b^2 \neq 0$. The elliptic curve $E(\mathbb{F}_p)$ is the set of points $(x, y) \in \mathbb{F}_p^2$ whose coordinates satisfy Equation 1. The rational points of E, augmented with a neutral element O called point at infinity, have an abelian group structure. The associated addition law computes the sum of two points in affine coordinates $P = (x_1, y_1)$ and $Q = (x_2, y_2)$ as $P + Q = (x_3, y_3)$ where: $x_3 = \lambda^2 - x_1 - x_2$ and $y_3 = \lambda(x_1 - x_3) - y_1$ with

$$\lambda = \begin{cases} \dfrac{y_2 - y_1}{x_2 - x_1} \text{ if } P \neq Q, \\ \dfrac{3x_1^2 + a}{2y_1} \text{ if } P = Q. \end{cases} \tag{2}$$

The scalar multiplication of a point P by a natural integer α is denoted αP. The discrete logarithm problem is finding the value of α, given P and αP. The security of ECC is based on the hardness of the discrete logarithm problem. Point addition formulae such as in Equation 2 are based on several operations over $\mathbb{F}_p$ (e.g. multiplication, inversion, addition, and subtraction) which have different computational costs.

ECC in Jacobian coordinates In Jacobian coordinates, we use $(x : y : z)$ to represent the affine point $(x/z^2; y/z^3)$. The elliptic curve equation becomes:

$$Y^2 = X^3 + aXZ^4 + bZ^6.$$

Doubling step: we represent the point $Q \in E(\mathbb{F}_p)$ in Jacobian coordinates as $Q = (X_Q, Y_Q, Z_Q)$. The formulae for doubling $T = 2Q = (X_T, Y_T, Z_T)$ can be computed as:

$$X_T = 9X_Q^4 - 8X_QY_Q^2. \quad Y_T = 3X_Q^2(4X_QY_Q - X_T) - 8Y_Q^4. \quad Z_T = 2Y_QZ_Q.$$

Adding step: let $Q = (X_Q, Y_Q, Z_Q)$ and $T = (X_T, Y_T, Z_T) \in E(\mathbb{F}_p)$. Then the point $R = T + Q = (X_R, Y_R, Z_R)$, can be computed as:

$$X_R = (2Y_QZ_T^3 - 2Y_T)^2 - 4(X_QZ_T^2 - X_T)^3 - 8(X_QZ_T^2 - X_T)^2X_T.$$
$$Y_R = (2Y_QZ_T^3 - 2Y_T)(4(X_QZ_T^2 - X_T)^2X_T - X_R) - 8Y_T(X_QZ_T^2 - XT)^3.$$
$$Z_R = 2Z_T(X_QZ_T^2 - X_T).$$

Algorithm 1: Scalar Multiplication

Input: $\alpha = (\alpha_n \alpha_{n-1} \dots \alpha_1 \alpha_0)_2$ radix 2 decomposition $\in \mathbb{F}_p$, $P \in E(\mathbb{F}_p)$
Output: αP

$T \leftarrow P$
for $i = n - 1$ *to* 0 **do**
  $T \leftarrow 2T$
  if $\alpha_i = 1$ **then**
    $T \leftarrow T + P$
return T

3 Our architecture for MMM (CIOS Method)

The Coarsely Integrated Operand Scanning (CIOS) method presented in Algorithm 2, improves the Montgomery Algorithm by integrating the multiplication and reduction. More specifically, instead of computing the product $a \cdot b$, then reducing the result, this method allows an alternation between iterations of the outer loops for multiplication and reduction. The integers (p, a and b) are seen as lists of s words of size w. This algorithm requires an array T of size only $s + 2$ to store the intermediate state. The final result of the CIOS algorithm is composed by the $s + 1$ least significant words of this array at the end.

Algorithm 2: CIOS algorithm for Montgomery multiplication [11]

Input: $p < 2^K$, $p' = -p^{-1} mod\ 2^w$, w, s , $K = s \cdot w$:bit length, $R = 2^K$, $a, b < p$
Output: $a \cdot b \cdot R^{-1}$ mod p

$T \leftarrow 0$;
for $i \leftarrow 0$ ***to*** $s - 1$ **do**
  $C \leftarrow 0$;
  for $j \leftarrow 0$ ***to*** $s - 1$ **do**
    $(C, S) \leftarrow T[j] + a[i] \cdot b[j] + C$ } α cell
    $T[j] \leftarrow S$
  $(C, S) \leftarrow T[s] + C$ } α_f cell
  $T[s] \leftarrow S$
  $T[s + 1] \leftarrow C$
  $m \leftarrow T[0] \cdot p' \bmod 2^w$ } β cell
  $(C, S) \leftarrow T[0] + m \cdot p[0]$
  for $j \leftarrow 1$ ***to*** $s - 1$ **do**
    $(C, S) \leftarrow T[j] + m \cdot p[j] + C$ } γ cell
    $T[j - 1] \leftarrow S$
  $(C, S) \leftarrow T[s] + C$ } γ_f cell
  $T[s - 1] \leftarrow S$
  $T[s] \leftarrow T[s + 1] + C$
return T;

The alternation between multiplications and reductions is possible since the value of m (in line 11 of the Algorithm 2) in the i^{th} iteration of the outer loop for reduction depends only on the value $T[0]$, which is computed by the first iteration of the corresponding inner loop. In order to perform the multiplication, we have modified the CIOS algorithm of [11] and designed this method with a systolic architecture. Indeed, instead of using an array to store the intermediate result, we replace T by input and output signals for each Processing Element. As a consequence, our design uses fewer multiplexers decreasing as a consequence the number of slices taken by our design.

4 Hardware Implementation

4.1 Block DSP in Xilinx FPGAs

Modern FPGA devices like Xilinx Virtex-4, Virtex-5 and Artix-7 as well as Altera Stratix FPGAs have been equipped with arithmetic hardcore extensions to accelerate digital signal processing applications. These DSP blocks can be used to build a more efficient implementation in terms of performance and reduce at the same time the demand for area. DSP blocks can be programmed to perform basic arithmetic functions, multiplication, addition and subtraction of unsigned integers. Figure 1 shows the generic DSP structure in advanced FPGAs. DSP can operate on external input A,B and C as well as on feedback values from P or result PCIN.

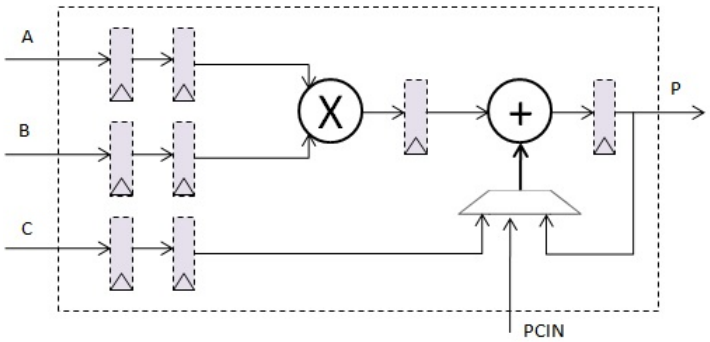

Figure 1: Structure of DSP block in modern FPGA device.

4.2 Proposed Architecture

The idea of our design is to combine the CIOS method for the MMM presented in [11] with a two-dimensional systolic architecture in the model of [13,20,24]. As seen in section 3, the CIOS method is an alternation between iterations of the loops for multiplication and reduction. The concept of the two-dimensional systolic architecture presented in Section 2 combines Processing Elements with local connections, which take external inputs and handle them with a predetermined manner in a pipelined fashion. This new architecture is directly based on the arithmetic operations of the CIOS method

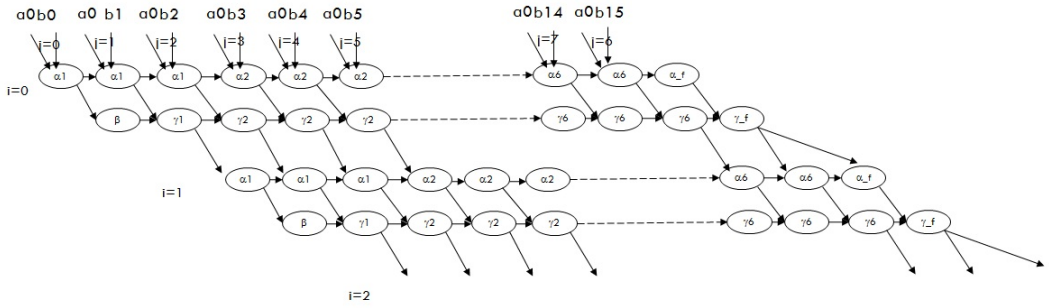

Figure 2: data dependency in general systolic architecture.

of Montgomery Algorithm. The arithmetic is performed in a radix-w base (2^w). The input operands are processed in s words of w bits. We present many versions of this method. We illustrate our design for $s = 16$ architecture, denoted NW-16 (for Number of Words). We describe it in detail as well as the various Processing Element behaviours. In order to have less states in our Final State Machine (FSM), we divided our Algorithm 2 of Montgomery in five kinds of PEs (α, β, γ, α_f, γ_f). Our efficient architecture comes from the fact that the data dependency in the CIOS algorithm allows to perform several operations in parallel. Figure 2 presents the dependency of the different cells. Below we describe precisely each cell. The letters MSW stand for the Most Significant Word and LSW for the Least Significant Word. In our notation the letter C denote the MSW of the result and the letter S the LSW.

α cell The α cell corresponds to lines 5 and 6 of Algorithm 2. Its operations are described in Algorithm 3. Notice how registers are embedded into the cell, avoiding in this manner

Algorithm 3: α cell

Input: $a[i]$, $b[j]$, $C_{In}(= C)$, $S_{In}(= T[j])$
Output: C_{Out}, S_{Out}
1 $t1 \leftarrow S_{In} + C_{In}$
2 $t2 \leftarrow a[i] \cdot b[j]$
3 $t3 \leftarrow t2 + t1$
4 $C_{Out} \leftarrow$MSW($t3$)
5 $S_{Out} \leftarrow$LSW($t3$)
6 **return** C_{Out}, S_{Out}

the usage of an external memory. This cell corresponds to one iteration of the first inner loop. As such it must be used several times in a row (the number of iterations in the inner loop). This chain of α cells is terminated by an α_f cell (alpha final) corresponding to lines 7, 8 and 9 of Algorithm 2. Once the first α cell (for each outer loop) is computed, data is available for the β cell (it requires $T[0]$). It is preferable to consume this data as soon as it is available as to minimize memory usage.

α_f cell The α_f cell corresponds to the end of the first inner loop. It is just an addition as shown in Algorithm 4.

Algorithm 4: α_f cell

Input: $C_{In}(= C)$, $S_{In}(= T[s])$
Output: $C_{Out}(= T[s+1])$, $S_{Out}(= T[s])$
1 $t1 \leftarrow S_{In} + C_{In}$
2 $C_{Out} \leftarrow$MSW($t1$)
3 $S_{Out} \leftarrow$LSW($t1$)
4 **return** C_{Out}, S_{Out}

β cell The β cell is used to compute the m value for each outer loop as well as the first special iteration of the second inner loop. Its operations are described in Algorithm 5. Once the β cell computation has been done, it now becomes possible to compute the

Algorithm 5: β cell

Input: $S_{In}(= T[0])$, p', $p[0]$
Output: C_{Out}, S_{Out}
1 $t1 \leftarrow S_{In} \cdot p'$
2 $m \leftarrow$LSW($t1$)
3 $t2 \leftarrow p[0] \cdot m$
4 $t3 \leftarrow S_{In} + t2$
5 $C_{Out} \leftarrow$MSW($t3$)
6 **return** C_{Out}, m

second inner loop with γ cells.

γ cell The γ cell corresponds to one iteration of the second inner loop. As such these cells must be chained so as to complete the whole second inner loop. For each cell in this chain,the same m value is required. Value that is changed for each outer loop iteration (m is computed by the β cell). Since a new m value may be computed before the end of the γ chain, the value must be propagated along the cells in the same chain. Its operations are described in Algorithm 6.

Algorithm 6: γ cell

Input: $C_{In}(= C)$, $S_{In}(= T[j])$, m, $p[j]$
Output: C_{Out}, S_{Out}
1 $t1 \leftarrow S_{In} + C_{In}$
2 $t2 \leftarrow m \cdot p[j]$
3 $t3 \leftarrow t1 + t2$
4 $C_{Out} \leftarrow$MSW($t3$)
5 $S_{Out} \leftarrow$LSW($t3$)
6 **return** C_{Out}, S_{Out}, m

γ_f *cell* Finally, the γ_f cell terminate each γ cell chain. It consists in two additions as shown in Algorithm 7. The difference with the α_f cell is that in this case both output

Algorithm 7: γ_f cell

Input: $C_{In}(= C)$, $S1_{In}(= T[s])$, $S2_{In}(= T[s+1])$
Output: $C_{Out}(= T[s])$, $S_{Out}(= T[s-1])$
1 $t1 \leftarrow S1_{In} + C_{In}$
2 $t2 \leftarrow$MSW($t1$)
3 $S_{Out} \leftarrow$LSW($t1$)
4 $C_{Out} \leftarrow$LSW($S2_{In} + t2$)
5 **return** $C_{Out}(= T[s])$, $S_{Out}(= T[s-1])$

values S and C are used in the rest of the computation. C is used by the α_f cell and S by an α cell.

4.3 Our architectures

Firstly, we will start with the NW-16 architecture which contains 6 PEs of type alpha and 6 of type gamma. An MMM can be performed with this architecture in 66 clock cycles. Similarly, in order to implement the NW-32 architecture and the NW-64 architecture it is required to double the number of cells each time. We provide a comparison of our architectures at the end of this section.

NW-16 Architecture In this architecture, the operands and the modulo are divided in 16 words. The NW-16 architecture is designed in the same way as the NW-32 an NW-64. This example illustrates the scalability of our design. The NW-16 architecture is composed of 15 Processing Elements distributed in a two-dimensional array, where every Processing Elements are responsible for the calculus involving w-bit words of the input operands.

The 15 Processing Elements are divided like this: 6 α cells, 1 α_f cell, 1 β cell, 6 γ cells et 1 γ_f cell. As said previously, the number of other PE type (α_f, β, γ_f) remains unchanged whatever the number of words in the design. In order to evaluate the number of clock cycles of the NW-16 architecture, the first parameter is

$$6 = \max\{\text{number of } \alpha \text{ cells}, \text{number of } \gamma \text{ cells}\},$$

implying that our algorithm requires to loop $s + 6$ times. We can perform the multiplication with our design in 66 clock cycles since our design requires three states ($66 = 3 \times (s + 6)$). The different results of this architecture for bit-length 256 are given in Table 1.

Architectures comparison The Table 2 shows a comparison between the different architectures. The number of clock cycles for every architecture is equal to $3 \times (s + nb)$, such that $nb = \max\{\text{number of } \alpha \text{ cells}, \text{number of } \gamma \text{ cells}\}$, implying that our algorithm

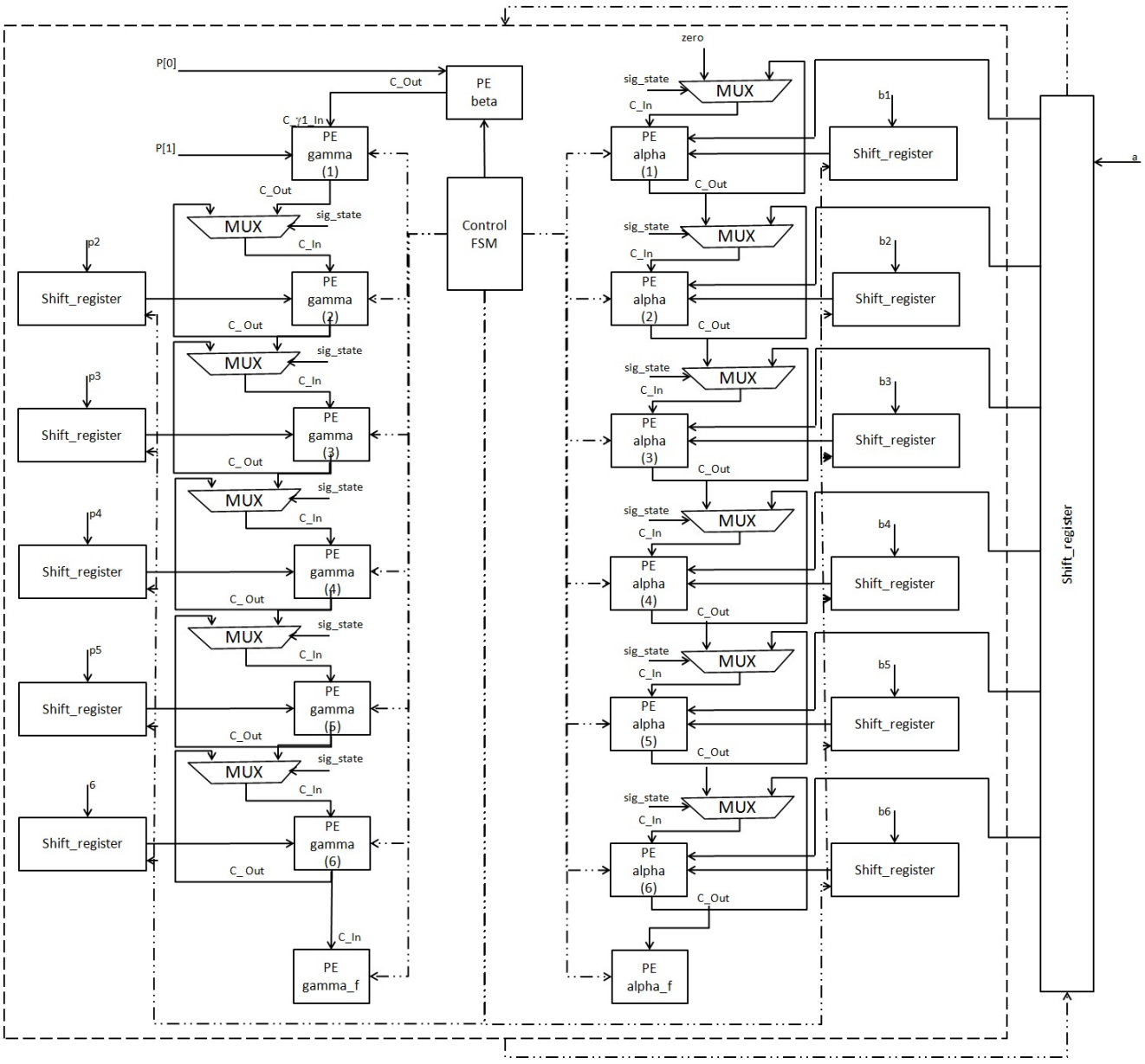

Figure 3: Proposed Montgomery modular multiplication architecture of NW-16.

Artix-7	DSP	Frequency	Clock cycles
MMM (s=16/K=256)	29	146	66
α	2	379	1
γ	2	379	1
β	2	453	1
α_f	1	460	1
γ_f	2	443	1

Table 1: Implementations of cells and MMM (NW-16).

require to loop $s + nb$ times. It is interesting to notice that all our architectures are scalable and can target the different security levels useful in cryptography.

CIOS	s=8	s=16	s=32	s=64
K=256	32	16	8	4
K=512	64	32	16	8
K=1024	128	64	32	16
K=2048	256	128	64	32
Clock cycles= $3 \times (s + nb)$	33	66	132	264
Number of cells	6 +3	12 +3	24 +3	48 +3

Table 2: Comparison of our architectures

4.4 ECC implementation

ECC algorithms when implemented in a sequential way have the advantage that the number of finite field arithmetic modules can be reduced to a minimum. For example, only one adder, one multiplier and one subtraction unit (can be an adder) are needed for point addition and doubling. Parallelization between multiplication, addition and subtraction was achieved. We proposed in Figure **??** and Figure **??** the dataflow graphs for point addition and doubling. In this design we use our efficient systolic architecture of

MMM to perform squaring or multiplication. The Table 4 summarizes the implementation results of the scalar multiplication. We present a results with both NW-8 and NW-16 architectures. To check the correctness of the hardware results, we compare the results given by the FPGA with sage software implementation.

5 Experimental Results

The Table 3 summarizes the FPGA post-implementation results for the proposed versions of MMM architectures. We present results for NW-16 architecture. The designs were described in a hardware description languages (VHDL) and synthesized for Artix-7 and Virtex-5 Xilinx FPGAs. We present the different results after implementation of bit-length k which are given in Table 3. As it is shown in Table 3, an interesting property of our design is the fact that the clock cycles are independent from the bit length. This property gives to our design the advantage of suitability to different security level. In order to implement the modular Montgomery multiplication for fixed security level, we must choose the most suitable architecture. The results presented in this work are compared with the previous work [4,5,18,19] in the Table 3. We can notice that our results are better than [19] considering every point of comparison i.e. the number of slice and the number of clock cycles. Considering the number of slices, we recall that [19] used an external memory to optimize the number of slices used by their algorithms. Considering the comparison with [18], our design requires less slices and can ran at a better frequency, without considering the huge progress in the number of clock cycles. Our design performed the MMM in 66 clock cycles for the 512 and 1024 bit length corresponding to AES-256 and AES-512 security level, while [18] performed the multiplication in 1540 clock cycles for the AES-256 security level and 3076 for the AES-512 security level.

6 Conclusion

In this paper we have presented an efficient hardware implementation of the CIOS method of MMM over large prime characteristic finite fields $\mathbb{F}_p$. We give the results of our design after routing and placement using a Artix-7 and Virtex-5 Xilinx FPGAs. Our systolic implementations is suitable for every implementation implying a modular multiplication, for example RSA, ECC and pairing-based cryptography. Our architectures and the designs were matched with features of the FPGAs. The NW-16 design presented a good performance considering $latency \times area$ efficiency. This architecture can run for all the bit length corresponding to classical security levels (128, 256, 512 or 1024 bits) in just 66 clock cycles. Our systolic design using this method CIOS is scalable for any other number of words. Then we showed that using this multiplier, it is possible to achieve an efficient scalar multiplication.

	Xilinx FPGAs													
	Our works A7		**Our works V5**		[19] V5		[18] VE		[5] VII	[4] VII	[14] V		[1] K7 and V5	
	512	**1024**	**512**	**1024**	512	1024	512	1024	1024	1024	512	1024	512 K7	512 V5
Freq	**106**	**65**	**97**	**65**	95	130	95.2	95.6	116.4	119	72.1	79.2	176	123
Cycles	**66**	**66**	**66**	**66**	96	384	1540	3076	1088	1167	–	–	66	66
Speed μs	**0.622**	**1.013**	**0.680**	**1.015**	1.010	2.953	16.031	32.021	9.34	9.80	–	–	0.374	0.536
Slice Reg	**2164**	**4208**	**3046**	**6072**	3876	6642	–	–	–	–	–	–	5076	4960
Slice LUTs	**1789**	**5242**	**1781**	**5824**	–	–	2972	5706	9319	9271	3125	6243	8757	10877
BRAM	0	0	0	0	128	256	–	–	–	–	–	–	0	0

Table 3: Comparaison of our work with state-of-art implementations.

	Slice	DSPs	BRAM	Freq	Slice FF	Slice LUT
NW-8 (256)	3745	33	12	98	8281	9722
NW-16 (256)	3770	34	12	130	8313	9255
NW-8 (512)	7066	92	23	59	16500	20394
NW-16 (512)	7116	60	23	74	16501	19199

Table 4: Implementations of ECC (Jacobian) in Xilinx FPGA.

References

[1] Bigou, K., Tisserand, A.: Single base modular multiplication for efficient hardware rns implementations of ecc. In: Conference on Cryptographic Hardware and Embedded Systems. p. 123–140 (September 2015)

[2] Hankerson, D., Menezes, A.J., Vanstone, S.: Guide to elliptic curve cryptography. Springer Science & Business Media (2006)

[3] Hariri, A., Reyhani-Masoleh, A.: Bit-serial and bit-parallel montgomery multiplication and squaring over gf. IEEE Transactions on Computers 58(10), 1332–1345 (2009)

[4] Harris, D., Krishnamurthy, R., Anders, M., Mathew, S., Hsu, S.: An improved unified scalable radix-2 montgomery multiplier. In: Computer Arithmetic, 2005. ARITH-17 2005. 17th IEEE Symposium on. pp. 172–178 (June 2005)

[5] Huang, M., Gaj, K., El-Ghazawi, T.: New hardware architectures for montgomery modular multiplication algorithm. Computers, IEEE Transactions on 60(7), 923–936 (July 2011)

[6] Huang, M., Gaj, K., Kwon, S., El-Ghazawi, T.: An optimized hardware architecture for the montgomery multiplication algorithm. In: Cramer, R. (ed.) Public Key Cryptography – PKC 2008, Lecture Notes in Computer Science, vol. 4939, pp. 214–228. Springer Berlin Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-78440-1_13

[7] Iwamura, K., Matsumoto, T., Imai, H.: Systolic-arrays for modular exponentiation using montgomery method. In: Rueppel, R. (ed.) Advances in Cryptology — EUROCRYPT' 92, Lecture Notes in Computer Science, vol. 658, pp. 477–481. Springer Berlin Heidelberg (1993), http://dx.doi.org/10.1007/3-540-47555-9_43

[8] Joux, A.: A one round protocol for tripartite diffie–hellman. Journal of Cryptology 17(4), 263–276 (2004), http://dx.doi.org/10.1007/s00145-004-0312-y

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

[10] Koblitz, N.: Elliptic curve cryptosystems. Mathematics of computation 48(177), 203–209 (1987)

[11] Koç, C., Acar, T., Kaliski, B.S., J.: Analyzing and comparing montgomery multiplication algorithms. Micro, IEEE 16(3), 26–33 (Jun 1996)

[12] Kung, H.: Why systolic architectures? Computer 15(1), 37–46 (Jan 1982)

[13] i Lee, K.: Algorithm and VLSI architecture design for H.264/AVC Inter Frame Coding. Ph.D. thesis, National Cheng Kung University, Tainan, Taiwan (2007)

[14] Manochehri, K., Pourmozafari, S., Sadeghian, B.: Montgomery and rns for rsa hardware implementation. Computing and Informatics 29(5), 849–880 (2012)

[15] Miller, V.: Use of elliptic curves in cryptography. In: Williams, H. (ed.) Advances in Cryptology — CRYPTO '85 Proceedings, Lecture Notes in Computer Science, vol. 218, pp. 417–426. Springer Berlin Heidelberg (1986), http://dx.doi.org/10.1007/3-540-39799-X_31

[16] Miller, V.S.: Use of elliptic curves in cryptography. In: Conference on the Theory and Application of Cryptographic Techniques. pp. 417–426. Springer (1985)
[17] Montgomery, P.L.: Modular multiplication without trial division. Mathematics of Computation 44(170), 519–521 (1985)
[18] Ors, S.B., Batina, L., Preneel, B., Vandewalle, J.: Hardware implementation of a montgomery modular multiplier in a systolic array. In: Parallel and Distributed Processing Symposium, 2003. Proceedings. International. pp. 8–pp. IEEE (2003)
[19] Perin, G., Mesquita, D.G., Martins, J.a.B.: Montgomery modular multiplication on reconfigurable hardware: Systolic versus multiplexed implementation. Int. J. Reconfig. Comput. 2011, 61–610 (Jan 2011), http://dx.doi.org/10.1155/2011/127147
[20] Reymond, G., Murillo, V.: A hardware pipelined architecture of a scalable montgomery modular multiplier over gf(2m). In: 2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig). pp. 1–6 (Dec 2013)
[21] Rivest, R., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21, 120–126 (1978)
[22] Rivest, R.L., Shamir, A., Adleman, L.: A method for obtaining digital signatures and public-key cryptosystems. Communications of the ACM 21(2), 120–126 (1978)
[23] Tenca, A.F., Koç, Ç.K.: A scalable architecture for montgomery multiplication. In: Koç, Ç.K., Paar, C. (eds.) Cryptographic Hardware and Embedded Systems, First International Workshop, CHES'99, Worcester, MA, USA, August 12-13, 1999, Proceedings. Lecture Notes in Computer Science, vol. 1717, pp. 94–108. Springer (1999), http://dx.doi.org/10.1007/3-540-48059-5_10
[24] Vucha, M., Rajawat, A.: Design and fpga implementation of systolic array architecture for matrix multiplication. International Journal of Computer Applications 26(3), 0975 – 8887 (july 2011)

A Elliptic curve operations scheduling

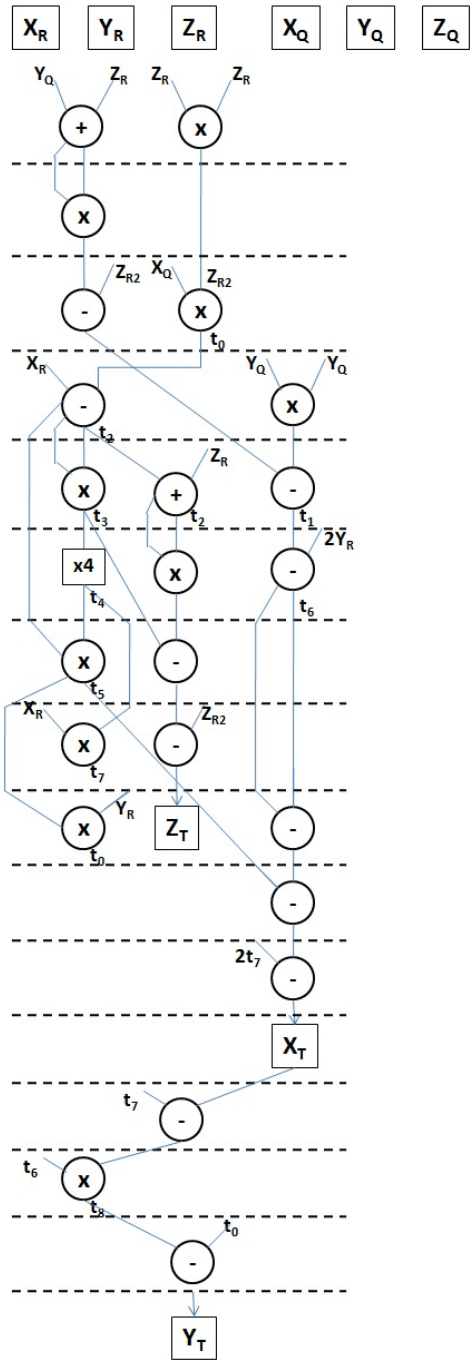

Figure 4: DFG for the point addition algorithm.

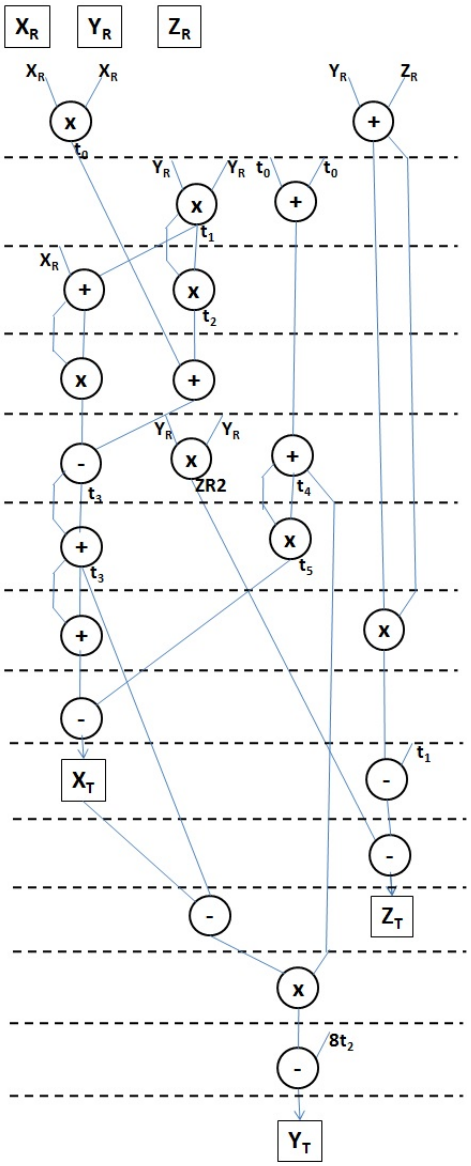

Figure 5: DFG for the point doubling algorithm.