Implementation of a pipelined modular multiplier architecture for GF(p) elliptic curve cryptography computation

ADNAN ABDUL-AZIZ GUTUB[*], ABDUL-RAHMAN M. EL-SHAFEI[**] AND MOHAMMED A. AABED[**]

[*]*Center of Research Excellence in Hajj and Omrah, Umm Al-Qura University, Makkah, Saudi Arabia. Associate Researcher, Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, Saudi Arabia.Email: aagutub@uqu.edu.sa*

[**]*Computer Engineering Department, King Fahd University of Petroleum & Minerals (KFUPM), Dhahran, Saudi Arabia*

Abstract

This paper reflects the achievement of improved results in terms of cost for Pipelined Crypto Modular Multiplier Architecture when compared with its earlier versions of Parallel Crypto Architecture. The improved pipelined modular multiplier is implemented on Field Programmable Gate Array (FPGA), designed in four stages to be properly suitable for elliptic curve crypto computation. To avoid inversion complexity, the elliptic computations arithmetic utilizes projective coordinates instead of the normal affine coordinates. We adjusted the elliptic curve crypto addition operation with efficient scheduling for this pipelining.

The proposed hardware is compared to the previous parallel (non-pipelined) models that were similarly designed. All considered architectures have been synthesized for 160-bits operations showing interesting features. Detailed results showed that this work gave overall efficiency in the cost, which shows a promising direction for further research.

Keywords: Cryptography hardware (Crypto Hardware); Elliptic curve cryptography (ECC); Parallel crypto architecture (Parallel Crypto Arch.); Pipelined crypto architecture (Pipelined Crypto Arch.); Projective coordinate cryptosystems (Proj. Crypto Sys.)

Introduction

Encryption used to be considered a very mysterious subject. It seemed to have no important practical applications outside the government/military world and was considered a harmless mathematical diversion if anyone else should pursue it. However, in today's world where everything we do is recorded on computers and all computers are linked to the Internet (and thus to each other), we find that encryption is of vital importance in many areas, such as the protection of privacy and confidentiality, transmission of secure information (e.g. credit card details), and to provide authentication of the sender of a message or even authenticate the time that message was sent. At the same time, law enforcement authorities worry that if everyone routinely kept their personal records encrypted, evidence held under a search warrant would be unusable. For this reason, they have tried to limit the ability of the general public to use strong (unbreakable) encryption. Therefore, the study of information security and cryptosystems has become an urgent need and a rich field of research and contribution. To cope with these challenges, one of the key areas has been the focus of research in the last two decades and is addressed in this work, Elliptic Curve Cryptography (ECC).

The study of elliptic curves by algebraists, algebraic geometers and number theorists dates back to the middle of the nineteenth century. The existence of an extensive literature describes the beautiful and elegant properties of these marvelous objects. In 1984, Hendrik Lenstra described an ingenious algorithm for factoring integers that relies on properties of elliptic curves (ECs). This discovery prompted researchers to investigate other applications of ECs in cryptography and computational number theory.

Public-key cryptography was conceived in 1976 by Diffie and Hellman. The first practical realization followed in 1977 when Rivest, Shamir and Adleman proposed their now well-known RSA cryptosystem, in which security is based on the intractability of the integer factorization problem. ECC was discovered for crypto usage in 1985 by Neal Koblitz and Victor Miller. Elliptic curve cryptographic schemes are public-key mechanisms that provide the same functionality as RSA schemes but with less computation. Over the years, researchers have developed techniques for designing and proving the security of RSA, Discrete Logarithm (DL) and EC protocols under reasonable assumptions. The fundamental security issue that remains is the hardness of the underlying mathematical problem that is necessary for the security of all protocols in a public-key family: the integer factorization problem for RSA systems, the discrete logarithm problem for DL systems, and the elliptic curve discrete logarithm problem for

ECC systems. The perceived difficulty of these problems directly impacts performance, since it dictates the sizes of the domain and key parameters. That in turn affects the performance of the underlying arithmetic operations. The security of these schemes is based on the difficulty of a different problem, namely the elliptic curve discrete logarithm problem (ECDLP). Currently the best algorithms known to solve the ECDLP have fully exponential running time, in contrast to the sub-exponential time algorithms known for the integer factorization problem. This means that a desired security level can be attained with significantly smaller keys in elliptic curve systems than is possible with their RSA counterparts. For example, it is generally accepted that a 160-bit elliptic curve key provides the same level of security as a 1024-bit RSA key. The advantages that can be gained from smaller key sizes include speed and efficient use of power, bandwidth, and storage.

ECC computations are performed in hardware as well as software, where hardware ECC computations have been proved to be faster and more efficient (Gutub 2006). This work proposes improving ECC hardware by pipelining its modular multiplier that is implemented on FPGA. We tune the used modular multiplier as a 4-stage pipeline suitable for elliptic curve crypto computation. To avoid inversion complexity, the elliptic computations arithmetic utilizes projective coordinates instead of the normal affine coordinates as proved by Gutub (2006), i.e. the proposed architecture considers representing the elliptic curve points as projective coordinate points in order to reduce the number of all inversion operations to one. We also adjusted the elliptic curve crypto addition operation with efficient scheduling for correct pipelining. Our proposed hardware is compared to parallel (non-pipelined) models designed very similarly to ensure fair comparisons and conclusions.

The flow of the paper will be as follows: first we provide some background behind the security of ECC and elliptic curve crypto computations. This section will also provide an example of simple procedure for using ECC for encryption and decryption. Several available techniques for implementing ECC at high-speed are explored in the section to follow. All ECC design schemes are to sustain the high throughput required by applications, where hardware-based designs are shown as the solution reaching acceptable performance-cost ratio. Section 4 describes the ECC scalar multiplication concept and algorithm, which is detailed and reconsidered within this work as coordinate systems (Section 5). Section 6 covers designing the hardware components used with the modules data flow that is improved for this architecture. The proposed pipelined design is detailed in Section 7. Section 8 elaborates in the pipelining stages derivation proofing the accuracy of the proposed design. The hardware implementation and simulation results are given in Section 9 followed by the comparison and analysis remarks (Section 10). Finally, Section 11 concludes the work.

Way to ECC

This section provides background behind the security of elliptic curve crypto computations. It also provides an example of a simple procedure for using ECC for encryption and decryption.

The cryptographic protocols generally serve very similar objectives and are based on almost same principles (Blake *et al.,* 1999). They contain a function which, by means of a parameter called the encryption key, can be easily computed. The inverse of this function is hard to compute unless a trapdoor function (a second key corresponding to the former one) is known. A general assumption made during the analysis of the security of a system is that all information about the system except the trapdoor key are known by the adversary. The previously mentioned group of public and private key systems is based on the way these keys are generated and kept.

In a private key system, encryption and decryption are performed using the same key which should be kept secret, otherwise the system is broken. In public key cryptosystems, encryption and decryption are done using two different keys. One of the keys is published and the other is kept secret. When one party is going to sign a message, the encryption key is kept secret but the key to verify the signature will be published. On the other hand, when a secret message is to be sent, the encryption key will be published while the key to open the message will be kept secret by the owner. For example, when messages sent to a user are encrypted by his public key, he is the only person who has access to the corresponding private key and can decrypt the message. There are several types of public key cryptosystems. A major group of these systems is based on the difficulty of solving the discrete logarithm problem, which is the basic security feature behind Elliptic Curve Cryptography (ECC).

ECC bases its security on the difficulty in solving the discrete logarithm problem (DLP). ECC's main operation can be simplified to the sum of elliptic curve points, i.e. the ECC operation is to compute point Q given point P, where $Q=kP$ and this kP is interpreted as the sum of points $P+P+.....+P+P$ (k-times). Here, the points P and Q should satisfy an elliptic curve equation defined on a finite field (a finite set of elements that satisfies some axioms). The logarithm discrete problem is now defined: Given the points P and Q, find the integer k such that $Q=kP$. This point summation operation (kP) is called scalar multiplication. The difficulty in solving this conjectured problem increases as the number of elements (valid points) increase.

There are many ways to apply elliptic curves for encryption/decryption purposes. In its most basic form, users randomly chose a base point (x,y) lying on the elliptic curve E. The plaintext (the original message to be encrypted) is coded into an elliptic curve point (x_m,y_m). Each user selects a private key 'n' and computes his public key $P=n(x,y)$. For example, user A's private key is n_A and his public key is $P_A=n_A(x,y)$. For anyone to encrypt and send the message point (x_m,y_m) to user A, he/she needs to choose a random integer k and generate the ciphertext $C_m=\{k(x,y),(x_m,y_m)+kP_A\}$. The ciphertext pair of points uses A's public key, wherein only user A can decrypt the plaintext using his private key. To decrypt the ciphertext C_m, the first point in the pair C_m, $k(x,y)$ is multiplied by A's private key to get the point: $n_A(k(x,y))$. When this point is subtracted from the second point of C_m, the result will be the plaintext point (x_m,y_m). The complete decryption operations are $[(x_m,y_m)+kP_A]-A[k(x,y)]= (x_m,y_m)+k[n_A(x,y)]- n_A[k(x,y)]= (x_m,y_m)$.

The most time consuming operation in the encryption and decryption procedure is finding the multiples of the base point (x,y). The approach used to implement this is discussed in the Section 4, after covering the literature review of Section 3.

Literature review

Elliptic Curve Cryptography (ECC) is gaining increased research acceptance and has lately been the subject of several standards (Tawalbeh *et al.*, 2010). This interest is mainly due to the high level of security with relatively small keys provided by ECC. To sustain the high throughput required by applications such as network servers, high speed implementations of public-key cryptosystems are needed. For that purpose, hardware-based processors or accelerators are often the optimum solution for reaching an acceptable performance-cost ratio. The fundamental question that arises is how to choose the appropriate efficiency–flexibility tradeoff. In this section, techniques for implementing ECC at a high speed are explored.

Some studies deal with architectures using only one field size and one hardwired irreducible polynomial (or modulo). This polynomial is of course inconsistent through reconfiguration of reconfigurable platforms. A fully hardwired asynchronous Applicaion-Specific Integrated Circuit (ASIC) design is presented by K. Ja'rvinen *et al.* (2004). It is based on a special Optimal Normal Basis (ONB) multiplier. Another similar ASIC implementation, based on synchronous polynomial basis multiplier, can be found in (Sozzani *et al.,* 2005). Designs for curves defined over GF(p) finite fields are explained in Batina *et al.*, (2004), McIvor *et al.* (2004), Orlando & Paar, (2001) and Ors *et al*.(2003), where all use Montgomery multiplications. Three architectures (Bantina *et al.,* 2004; Orlando & Paar, 2001; Ors *et al*., 2003) use systolic array Montgomery modular multipliers while the design in McIvor *et al.,* (2004) uses Montgomery multiplication based on parallel schoolbook multipliers. A systolic array Montgomery modular multiplier is also modeled by N. Mentens *et al.* (2004) but for GF(2^m) curves. Other research work using reconfigurable hardware can be found in Bajracharya *et al.* (2004) and Nguyen *et al.,* (2003). They focus on an efficient hardware/software partition for the scalar multiplication. Reconfigurable architectures based on digit-serial polynomial basis multiplier can be found in Ansari & Hasan (2006), Lutz & Hasan (2004), Orlando & Paar (2000) and Shu *et al.,* (2005). The research by Ansari & Hasan (2006) also mentions ASIC results performing the fastest implementation. Saqib (2004) presented utilizing Hessian elliptic curves from Smart (2001) but they were used in order to extract more parallelism. The same design using the DLP is presented by Saqib *et al.* (2004). Nevertheless, the low frequency of this design shows that large multipliers must be handled with special care. More about Karatsuba multipliers can be found in Dyka & Langendoerfer (2005), Gathen *et al.,* (2005) and Rodríguez-Henríquez & Koç (2003). Considering several designs, the hardware shown by Ansari & Hasan (2006) seems to be the best representative of a high-speed architecture using a hardwired irreducible polynomial, a single field size, and an unknown base point *P*. It is a pipelined and parallel structure based on a large Most Significant Digit (MSD)-first multiplier and an accurate scheduling of the DLP algorithm.

The Sakiyama *et al.* (2007) work presents a reconfigurable curve-based crypto-processor that accelerates scalar multiplication of ECC and Hyper Elliptic Curve Cryptography (HECC) of genus 2 over GF(2^m). By allocating several copies *(*say *x)* of processing cores that embed reconfigurable Modular Arithmetic Logic Units (MALUs) over GF(2^m), scalar multiplication of ECC/HECC can be accelerated by exploiting Instruction-Level Parallelism (ILP). The supported field size can be arbitrary up to a limit related to the number of cores, i.e. up to $x(n+1)-1$. The super-scaling feature is facilitated by defining a single instruction that can be used for all field operations and point/divisor operations. Chen *et al.* (2007) presented a high-performance elliptic curve cryptographic processor for GF(p) general curves which featured a systolic arithmetic unit. The work proposed a new unified systolic array that efficiently implements addition, subtraction, multiplication and division. At the system level, the control dependencies in the operation sequence and the mismatched communication between the systolic array and the separate storage would stall the pipeline in the systolic array. The design avoided these pipeline stalls successfully using optimization methods. The processor is synthesized in 0.13-micron standard-cell technology. It required 1.01-ms to compute a 256-bit scalar multiplication for general curves over GF(p).

Kumar *et al*. (2006) showed different architectural enhancements in a Least Significant Digit (LSD) multiplier for binary fields GF(2^m). They proposed two different architectures: the Double Accumulator Multiplier (DAM), and N-Accumulator Multiplier (NAM), which are faster than traditional LSD multipliers. The evaluation of the multipliers for different digit sizes gave best choices and showed that currently used digit sizes are not efficient. The paper suggested that one should always use the NAM architecture to get the best timings. Considering the time area product, DAM or NAM gave the best performance depending on the digit size.

A GF(p) processor suitable for RSA, DSA and ECDSA is presented by K. Itoh *et al.* (1999). It is based on a Montgomery multiplier with a 16-bit digit size, implemented in a TMS320C6201 DSP (digital signal processor) hardware. Another processor using MAC2424 (24x24-bit) for GF(p^{10}) OEF is presented by Chung *et al.* (2000). It includes an interesting analysis of the pre-computation needed by the signed window scalar multiplication method. More about GF(p^m) arithmetic can be found in Bertoni *et al.* (2003) and Bajard *et al.* (2003). Two 64-bit dual-field processors can be found in Eberle *et al.* (2004) and Satoh & Takano (2003). They also use Montgomery multipliers and are both able to compute integer and binary polynomial arithmetic (for GF(p) and GF(2^m)). In particular, RSA, DSA, ECDSA and DH are supported. The architecture proposed in Satoh & Takano (2003) is an ASIC, while Eberle *et al.* (2004) is implemented on FPGA with ASIC extrapolations. The work in Eberle *et al.* described an architecture based on a 64-bit datapath with its core based on a 64x64-bit multiplier able to process both integer and binary polynomial arithmetic. The processor currently implements the main public-key protocols like RSA, DSA, ECDSA and DH with arbitrary key sizes and curves.

Scalar multiplication

The ECC algorithm used for calculating nP from P is based on the binary method. The algorithm used for scalar multiplication is based on the binary method (Gutub, 2006; Gutub & Ibrahim, 2003), since it is efficient for hardware implementation. The binary method algorithm is shown below:

```
Inputs:  $k$: a constant , P: point on the elliptic curve
Output: $Q$: another point on the elliptic curve, $Q=k . P$
Define: $w$: number of bits in $k$, where $k_i$ is the $i^{th}$ bit in $k$
-----------------------------------------------------------
If $k_{w-1}$:=1 then $Q := P$ else $Q := 0$;
for $i := w-2$ down to 0 do
    $Q := Q + Q$;    Point Doubling
    If $k_{w-1}=1$ then  $Q := Q + P$;    Point Addition
Return $Q$;
```

Basically, the binary method algorithm scans the bits of n and doubles the point Q k-times. Whenever a particular bit of n is found to be one, an extra operation is needed. This extra operation is $Q+P$. As a result, adding and doubling elliptic curve points are the most basic operations in each iteration. The point addition and doubling requires performing inversion operations. It is known that these inversion operations are the most expensive operations since their cost is equivalent to 9~30 modular multiplications (Gutub & Ibrahim, 2003); i.e., inversion is extremely slow, which made researchers generally try to avoid it if possible. The use of coordinate systems other than the Affine coordinate system (which will be illustrated later) greatly reduces the number of inversions required in the operations of the scalar multiplication on the expense of extra multiplications.

ECC effectively uses point doubling and addition operations in the arithmetic execution. The optimized formulae available for these operations eliminate the costly field inversion from the main loop of the scalar multiplication; i.e. fast operations are achieved by using projective coordinates (Gutub, 2006). However, the operation in projective coordinates involves more scalar multiplication than in affine coordinate and ECC on projective coordinates, and will be efficient only when the implementation of scalar multiplication is much faster than a multiplicative inverse operation. Therefore, transfer is needed from one coordinate to another for avoiding the inversion process cost. The following section is dedicated to illustration of the coordinate systems structure used for these purposes.

Coordinate systems

An elliptic curve can be represented by different coordinate systems. Following are descriptions of two coordinates, i.e. affine coordinate and standard projective coordinate procedures (Miyaji, 1992). This standard projective coordinate system is found appropriate for parallel implementation, as detailed in Gutub & Ibrahim (2003), which led to our choice for this proposed pipelined hardware.

Affine coordinate:

Let **E** be an elliptic curve over GF(p), has the following equation:

E: $y^2=x^3+ax+b$ (mod p),

where a and b are constants satisfying $4a^3+27b^2\neq 0 \pmod p$.

Let $P=(x_1,y_1)$, $Q=(x_2,y_2)$, and $P+Q=(x_3,y_3)$, be points of E(GF(p)) ,

1 Addition formula: $x_3=\lambda^2-x_1-x_2$, $y_3=\lambda(x_1-x_3)-y_1$, where $\lambda=(y_2-y_1)/(x_1-x_2)$

2 Doubling formula: $x_3=\lambda^2-2x_1$, $y_3=\lambda(x_1-x_3)-y_1$, where $\lambda=(3x_1^2+a)/(2y_1)$

Addition time = 1 Inversion + 1 Squarings + 2 Multiplications + 6 Subtractions
Doubling time = 1 Inversion + 2 Squarings + 2 Multiplications + 1 Addition + 3 Subtractions

<u>Standard projective coordinate:</u>

For standard projective coordinates, we set $x=X/Y$ and $y=Y/Z$, giving the equation:

E: $Y^2Z=X^3+aXZ^2+bZ^3 (mod\ p)$,

Let $P=(X_1,Y_1,Z_1)$, $Q=(X_2,Y_2,Z_2)$ *and* $P+Q=(X_3,Y_3,Z_3)$ be points of E(GF(p)) ,

1 Addition formula: $X_3= vA$, $Y_3= u(v^2X_1Z_2 – A) – v^3Y_1Z_2$, $Z_3=v^3Z_1Z_2$
where, $u = Y_2Z_1 – Y_1Z_2$, $v = X_2Z_1 – X_1Z_2$, $A = u^2Z_1Z_2 – v^3 – 2v^2Y_1Z_2$

2 Doubling formula: $X_3= 2hs$, $Y_3= w(4B-h) – 8s^2Y_1^2$, $Z_3=8s^3$
where, $w = aZ_1^2+3X_1^2$, $s = Y_1Z_1$, $B = X_1Y_1s$, $h = w^2-8B$

Addition time = 12 Multiplications + 4 Subtractions

Doubling time = 7 Multiplications + 1 Addition + 1 Subtraction

The work in Gutub & Ibrahim (2003) proposed running these ECC point operations with projective coordinates on hardware with parallel multipliers. It was found that four parallel multipliers would give the maximum parallelization with the least number of multiplication steps as proven in Tawalbeh *et al.* (2010). We, in this work, are concentrating on the implementation of the ECC point addition operation for comparison purposes. Figure 1 shows a rearranged ECC point addition data flow graph assuming 4 parallel multipliers. The computation through this data flow graph (Figure 1) requires 4 steps of modular multipliers and 4 steps of modular adders.

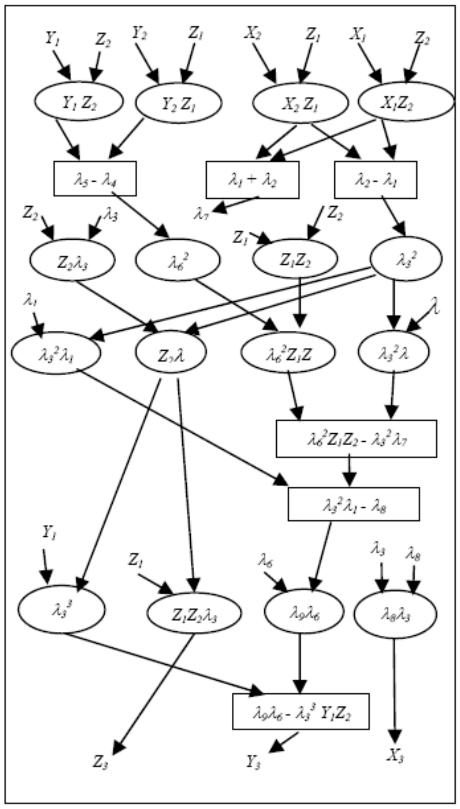

Fig. 1: Projecting (X,Y) to (X/Z,Y/Z) adding two points data flow.

Related hardware components

This section introduces the designs and algorithms considered for studying our pipelined hardware. We give a brief idea about modular addition first, followed by modular multiplication. Figure 2 shows the design implemented for modular addition operations in this model, extracted from the previous work by Gutub (2006).

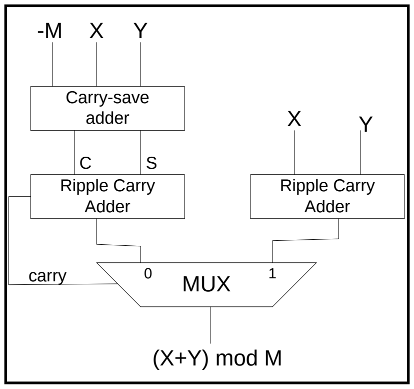

Fig. 2: Non-pipelined modular adder.

To obtain a pipelined implementation of this adder, it has been divided into two stages, as shown in Figure 3. The Ripple Carry Adders (RCAs) are split into two modules, each of which is a separate stage. The two RCAs in Figure 3 are divided into two stages using latches as shown in Figure 4. Note that this block diagram is a simplified example of a 6-bit operation.

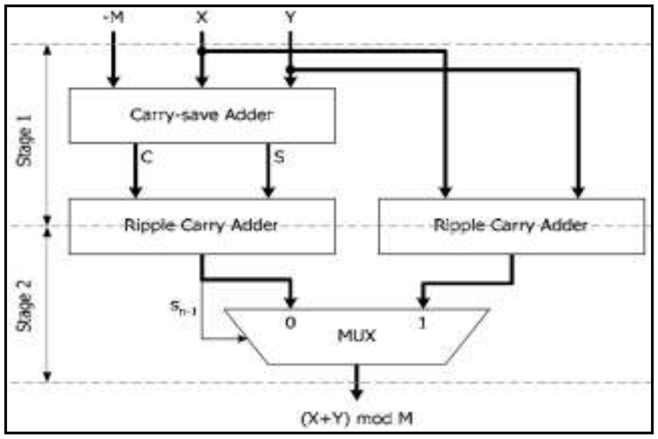

Fig. 3: Modular adder pipelined into two stages

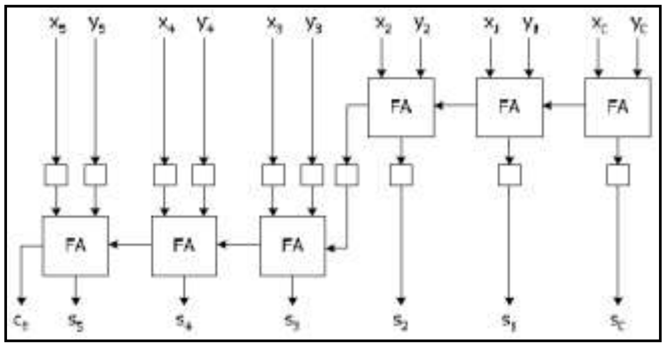

Fig. 4: 6-bit Ripple Carry Adder divided into two stages

As was mentioned earlier, the multiplication process is the most sophisticated and time consuming process in the ECC systems. Thus, optimizing the multiplier design and delay is a fundamental requirement for system efficiency. The straightforward approach to compute modular multiplication is by performing multiplication followed by reduction (Gutub, 2007). The multiplication can be computed through several addition operations. Then, the reduction is performed through several subtractions, by subtracting the modulus several times, until the result is less than the modulus. This approach is inefficient and suffers from very low speed. It can, however, be improved by merging modulo subtraction with the multiplication-add operations (Gutub, 2007), as in the algorithm below.

```
Define   k: number of bits in x;     $x_i$: the $i^{th}$ bit of x
Input:   x,y, and n; where x,y < n;
Output:  P = xy mod n

1.    P := 0;
2.    For i = k-1 down to 0;
3.        {
4.        P:= 2P;
5.        If $P \geq n$ Then P:= P – n ;
6.        If $x_i = 1$ Then
7.            {P := P + y;
8.            If $P \geq n$ Then P:= P – n};
9.        }
10.   End;
```

The algorithm above is for GF(p) modulo multiplication and found to be very appropriate for hardware implementation (Gutub, 2007). It has a bounding ‘for’ loop, which includes iterative modulo multiplication reduction operations. The bounding loop can be designed in hardware as a controller that will control the number and processes of the iterations. The modulo multiplication reduction is implemented in hardware with three adders and three multiplexers connected, as shown in Figure 5. There are no registers in the hardware design; the small boxes shown are symbols to clarify the mapping of bit-flow. The adder can function as a subtractor by inverting one of its inputs. The complete process of *x.y mod n* will need *k* clock cycles, if each modulo reduction iteration is performed in one clock cycle. The multiplication of *P* by two (as in step 4 of the algorithm above) is performed by a shift to the bits of *P* toward the left. The multiplexers Mux-1 and Mux-3 are controlled by the subtractor’s output-carry-bit. Therefore, the complete subtractions are to be made for the Mux to give the output. The reader is referred to Gutub (2007) for more details.

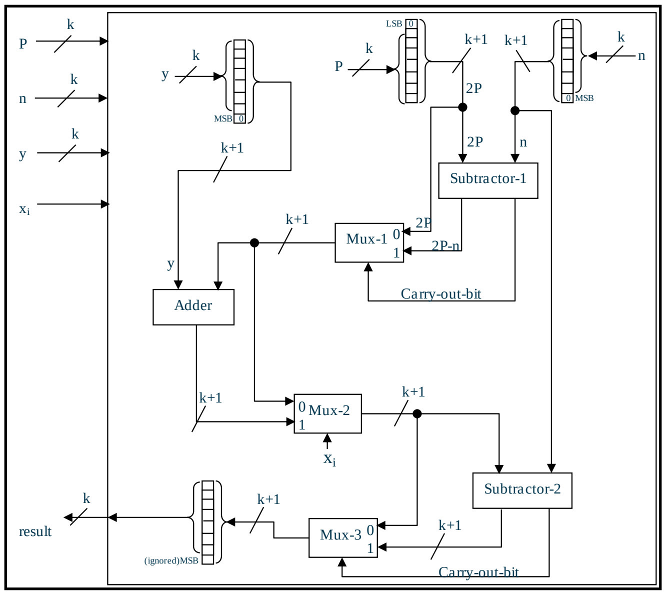

Fig. 5: Modular non-pipelined multiplier.

Proposed pipelined multiplier design

We propose improving the previous ECC design in Figure 6, detailed by Gutub & Ibrahim (2003), into a new 4-stage pipelined modular multiplication approach as shown in Figure 7. The design is similar in principle to our previous pipelined design in Gutub (2006); however, its architecture components are designed based on pipelining the standard projective coordinates which makes it an improved hardware with multipliers of four new stages. Each stage contains a different modular multiplication operation. Each multiplication operation loops through all stages *k*-times.

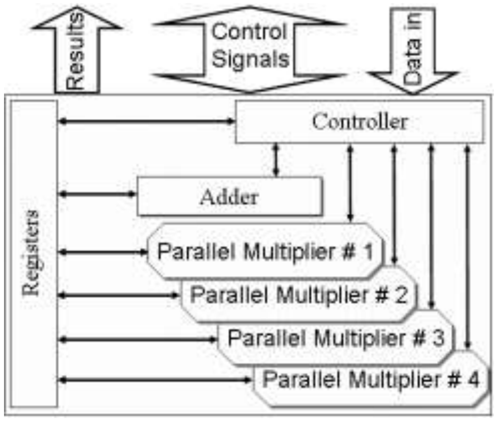

Fig. 6: Previous parallel architecture.

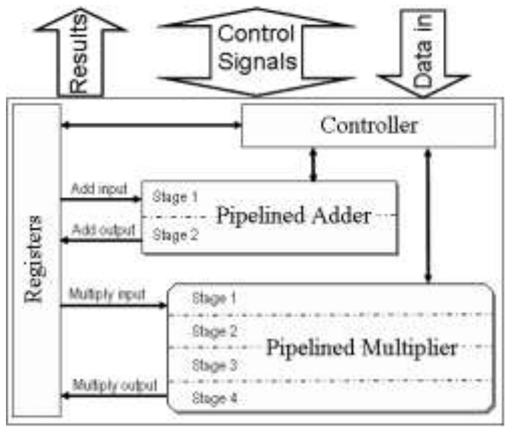

Fig. 7: Proposed pipelined architecture.

The modular adder consists of four *k/2* digit carry-propagate adder and three (*k/2+1*) digits carry-propagate adder. The modular multiplication can accommodate a maximum of 4 different multiplication stages (Figure 8), where each multiplication can be processed independently from other multiplications timings. However, new modular multiplication does not start looping through the pipeline until stage 1 is free. The output is delivered from stage 4 after *4k* cycles, as shows in Figure 8.

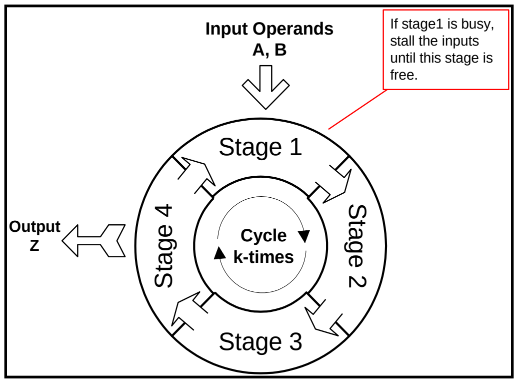

Fig. 8: Pipelined modular multiplier staged cycle

It should be noted that the control box is used to select when a new modular multiplication operation will be inserted. It also controls the looping of existing operations to complete *4k* cycles. The control box will allow a new multiplication operation if no current multiplication operation occupies this stage. If all stages are busy, the inputs will be stalled. Figure 9 shows the design implemented for the pipelined modular multiplication operations in our proposed model.

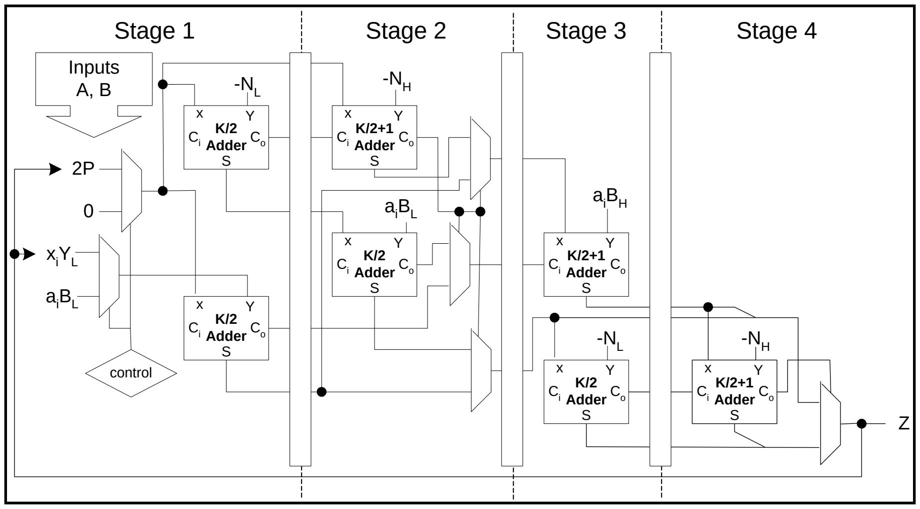

Fig. 9: Modular pipelined multiplier.

Figure 9 shows the four stages and where each stage needs to finalize its outputs for the next stage to start its operation accurately. Stage 1 processes the inputs at the first cycle; it contains two adders connected through multiplexers to direct the data correctly, i.e. either from as new inputs or as fed back from Stage 4 as an intermediate result. Note that Stages 2, 3, and 4 are also made up of adders that work together as a pipeline; they are following the k-times cycle shown in Figure 8 until the output is ready and to be taken from connection Z at Stage 4.

Derivation of pipelined point addition framework

The procedure for standard projective coordinates point addition can be defined corresponding to the registers as detailed in the following register sequence:

$R_1=Y_1Z_2$	$R_2=Y_2Z_1$	$R_3=X_1Z_2$	$R_4=Z_1X_2$	$R_5=R_2-R_1$	$R_6=R_3+R_4$
$R_7=R_4-R_3$	$R_8=R_7Z_2$	$R_9=R_5R_5$	$R_{10}=R_7R_7$	$R_{11}=Z_1Z_2$	$R_{12}=R_4R_{10}$
$R_{13}=R_8R_{10}$	$R_{14}=R_{11}R_9$	$R_{15}=R_6R_{10}$	$R_{16}=R_{14}-R_{15}$	$R_{17}=R_{12}-R_{16}$	$R_{18}=Y_1R_{13}$
$R_{19}= Z_3 =Z_1R_{13}$	$R_{20}=R_5R_{17}$	$R_{21}= X_3 =R_{16}R_7$	$R_{22}= Y_3 =R_{20}-R_{18}$		

The above procedure is rescheduled to avoid any dependency. It considers proper arrangement for trying to fully utilize all of the 4-stage modular pipelined multiplier (Figure 9). This rescheduling is shown in Table 1, which notes the number of clock cycles in an accumulation manner.

Table 1. ECC point addition scheduling for 4-stage pipelining.

Clock Cycle Accumulation	Modular Multiplication				Modular Adder		
	Stage 1	Stage 2	Stage 3	Stage 4	Stage 1	Stage 2	Comments
0	R_1						
1N	R_2	R_1					
2N	R_3	R_2	R_1				
3N	R_4	R_3	R_2	R_1			
4N	R_{11}	R_4	R_3	R_2			
4N+1		R_{11}	R_4	R_3	R_5		
4N+2			R_{11}	R_4		R_5	
4N+3	R_9			R_{11}	R_7		
4N+4	R_{11}	R_9			R_6	R_7	
4N+5	R_{10}	R_{11}	R_9			R_6	
4N+6	R_9	R_{10}	R_{11}	R_9			
8N		R_9	R_9	R_{10}			
8N+3	R_{14}	R_9	R_{10}				
8N+5	R_{15}		R_{14}	R_9			
8N+6	R_{12}	R_{15}		R_{14}			
8N+7	R_{14}	R_{12}	R_{15}				R_{13} is stalled
8N+8	R_{13}	R_{14}	R_{12}	R_{15}			
12N+3		R_{12}	R_{15}	R_{13}			
12N+5		R_{13}		R_{12}	R_{16}		
12N+6			R_{13}			R_{16}	
12N+7	R_{21}				R_{17}		
12N+8	R_{19}	R_{21}				R_{17}	
12N+9	R_{20}	R_{19}	R_{21}				
12N+10	R_{19}	R_{20}	R_{19}	R_{21}			Output X_3
16N+7		R_{19}	R_{20}	R_{19}			
16N+8			R_{19}	R_{20}			
16N+9				R_{19}	R_{22}		Output Z_3
16N+10						R_{22}	Output Y_3

In fact, this gives the clear estimation of the total number of clock cycles needed for an ECC point addition operation as *16N+10,* where *N* is the size of the modulus in bits.

Hardware implementations and simulations

The purpose of the hardware implementation is to give some common platform and fair comparison between our proposed pipelined architecture and similar previous designs.

The focus in this study is not targeted toward industrial purposes. It does not give the details of the architecture implementation; instead, the aim is to extract the hardware time and area parameters of the main blocks to build a fair comparison study between the designs. Therefore, our implementation exploration here is going to be limited to the level needed to serve this comparison goal. We will implement the basic blocks of hardware that are commonly used to build all studied designs, i.e. our model here as well as similar previous architectures. The major common components needed by all designs are the modular multiplier and modular adder.

For simulation, we have used Silos Compiler (free Verilog compiler) to generate the results. A sample of the output results are shown for each design implemented in the project. For synthesis, we used the Virtex-4 XC4VSX35 FPGA library of Xilinx IES software. Unfortunately, this available FPGA platform does not create an Area report, unlike synthesis using ASIC libraries. Note that the ASIC area report advantage gives an accurate estimate of the space requirements in micrometer^2 and number of gates needed. However, as mentioned before, this FPGA implementation cannot be precise for industrial usages; it is mainly a practical tool for fair comparison and academic study. We used it for comparing pipelined and non-pipelined versions of the ECC operations hardware. In this section, we have included a brief summary of the synthesis results (Table 2) followed by some output report-briefing subsections for each design.

Table 2. FPGA synthesis summary.

		Modular Adder		**Modular Multiplier**	
		Parallel	**Pipelined**	**Parallel**	**Pipelined**
Time	**Path delay**	8.36ns	7.4ns	6.05ns	3.2ns
	Frequency	119.6Mhz	135.2MHz	165.24MHz	312.4MHz
FPGA Hardware	**Slices**	18	29	59	105
	Flip Flops	-	28	33	139
	LUTs	32	48	105	141
	IOBs	32	36	36	36
This study Hardware Area Estimation Figure		82	141	233	421

Parallel Design (Non-pipelined) Modular Adder

- <u>Synthesis Results</u>

(a) Time:

The maximum combinational path delay is *8.359ns*. This is equivalent to having a maximum frequency of *119.6MHz*.

(b) FPGA Hardware:

- Number of Slices: 18 out of 15360 (0%).
- Number of 4 input LUTs: 32 out of 30720 (0%).
- Number of bonded IOBs: 32 out of 448 (7%).

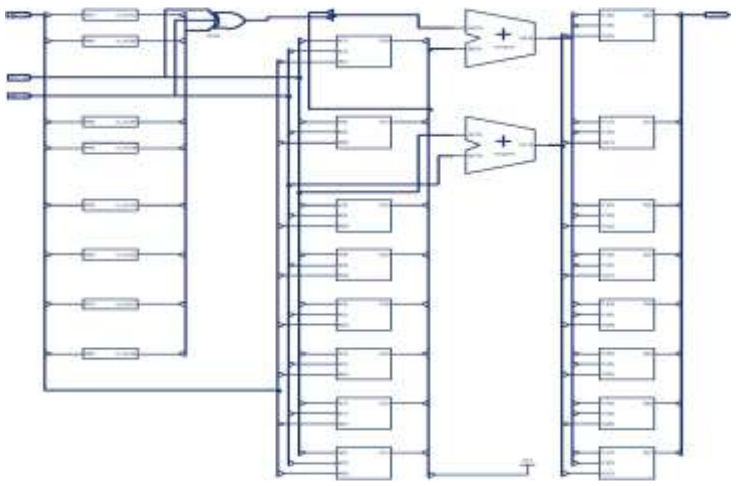

Fig. 10: Non-Pipelined Modular Adder Block Diagram.

▪ Macro Statistics

- Adders/Subtractors: 2
- 8-bit adder: 1
- 9-bit adder: 1
- XORs: 8
- 1-bit XOR3: 8

Pipelined Modular Adder

▪ Synthesis Results

(a) Time:

The minimum input arrival time before clock is *4.190ns*. The maximum output required time after clock is *7.399ns*. The maximum combinational path delay was not found. This is equivalent to having a maximum frequency of *135.2MHz*.

(b) FPGA Hardware:

- Number of Slices: 29 out of 15360 (0%)
- Number of Slice Flip Flops: 28 out of 30720 (0%)
- Number of 4 input LUTs: 48 out of 30720 (0%)
- Number of bonded IOBs: 36 out of 448 (8%)

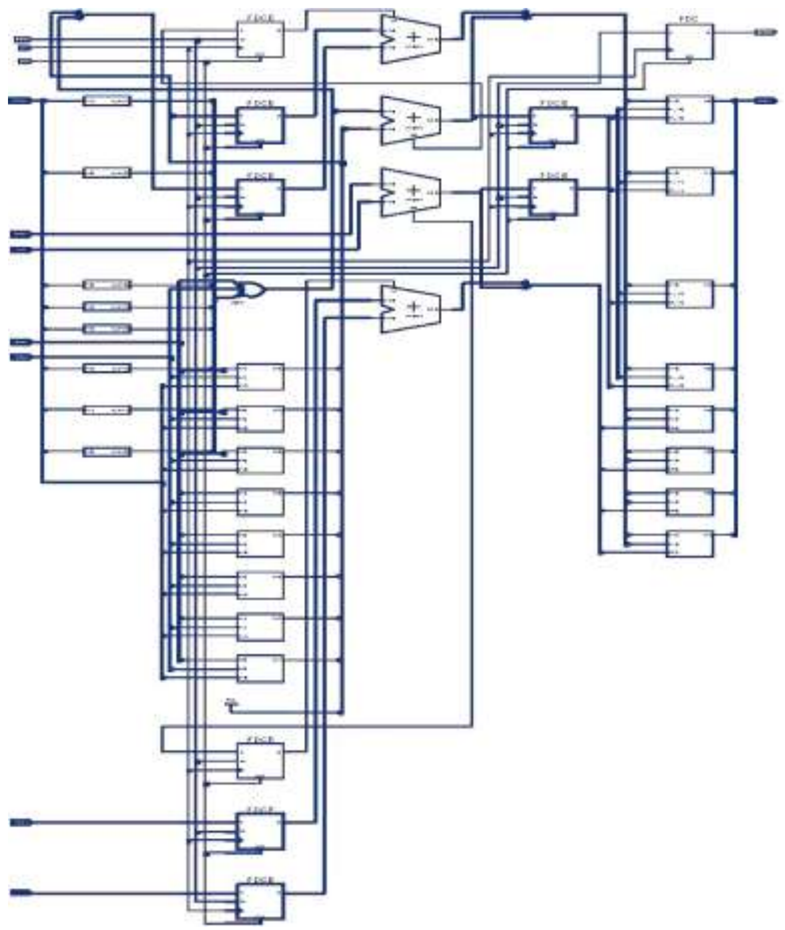

Fig. 11: Pipelined Modular Adder Block Diagram.

- Macro Statistics

- Adders/Subtractors: 4
- 4-bit adder carry in: 1
- 4-bit adder carry out: 2
- 5-bit adder carry in: 1
- Registers: 29
- Flip-Flops: 29
- XORs: 8
- 1-bit XOR3: 8

Parallel Design (Non-pipelined) Modular Multiplier

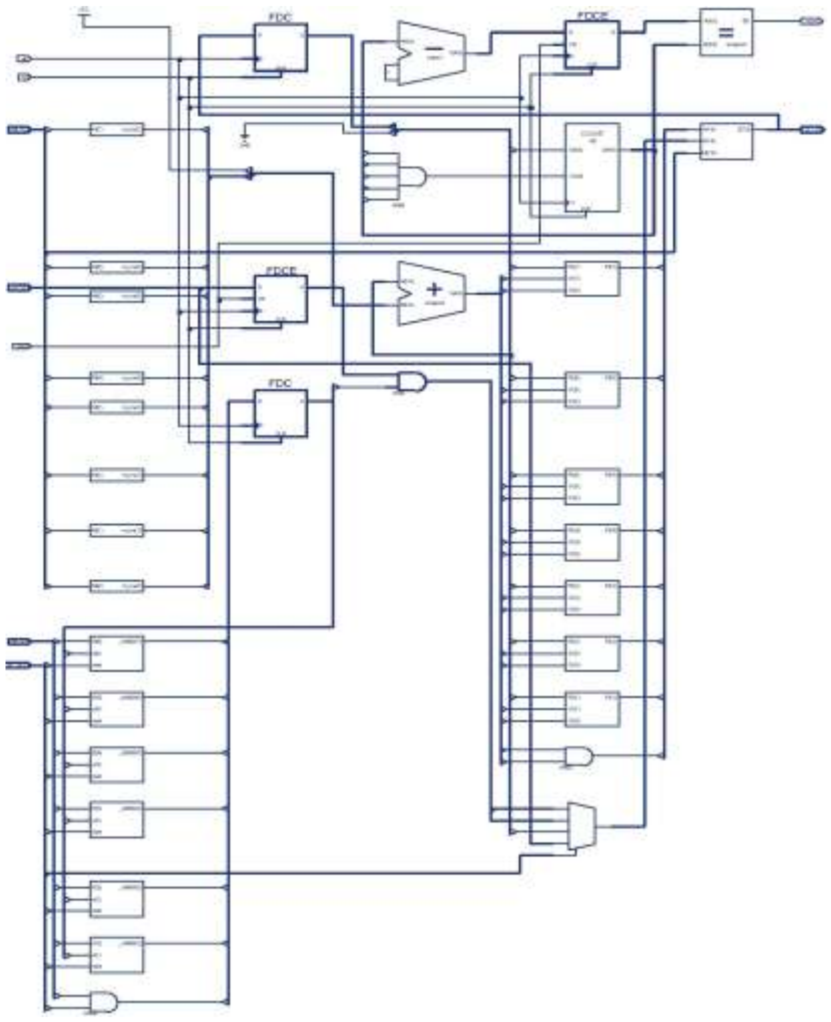

Fig. 12: Non-Pipelined Modular Multiplier Block Diagram.

- <u>Synthesis Results</u>

(a) Time:

The minimum period is *6.052ns* (Maximum Frequency: *165.238MHz*). The minimum input arrival time before clock is *6.633ns*. The maximum output required time after clock is *9.784ns*. The maximum combinational path delay is *10.365ns*.

(b) FPGA Hardware:

- Number of Slices: 59 out of 15360 (0%)
- Number of Slice Flip Flops: 33 out of 30720 (0%)
- Number of 4 input LUTs: 105 out of 30720 (0%)
- Number of bonded IOBs: 36 out of 448 (8%)

▪ <u>Macro Statistics</u>

- Adders/Subtractors: 4
- 5-bit subtractor: 1
- 8-bit adder: 1
- 9-bit adder: 2
- Counters : 1
- 5-bit up counter: 1
- Registers: 28
- Flip-Flops: 28
- Comparators: 1
- 5-bit comparator equal: 1
- Multiplexers: 1
- 8-bit 4-to-1 multiplexer: 1
- XORs: 8
- 1-bit XOR3: 8

Pipelined Modular Multiplier

▪ <u>Synthesis Results</u>

(a) Time:

The minimum period: *3.201ns* (Maximum Frequency is *312.402MHz*). The minimum input arrival time before clock is *3.904ns*. The maximum output required time after clock is *6.909ns*. The maximum combinational path delay is *7.629ns*

(b) FPGA Hardware:

- Number of Slices: 105 out of 15360 (0%)
- Number of Slice Flip Flops: 139 out of 30720 (0%)
- Number of 4 input LUTs: 141 out of 30720 (0%)
- Number of bonded IOBs: 36 out of 448 (8%)

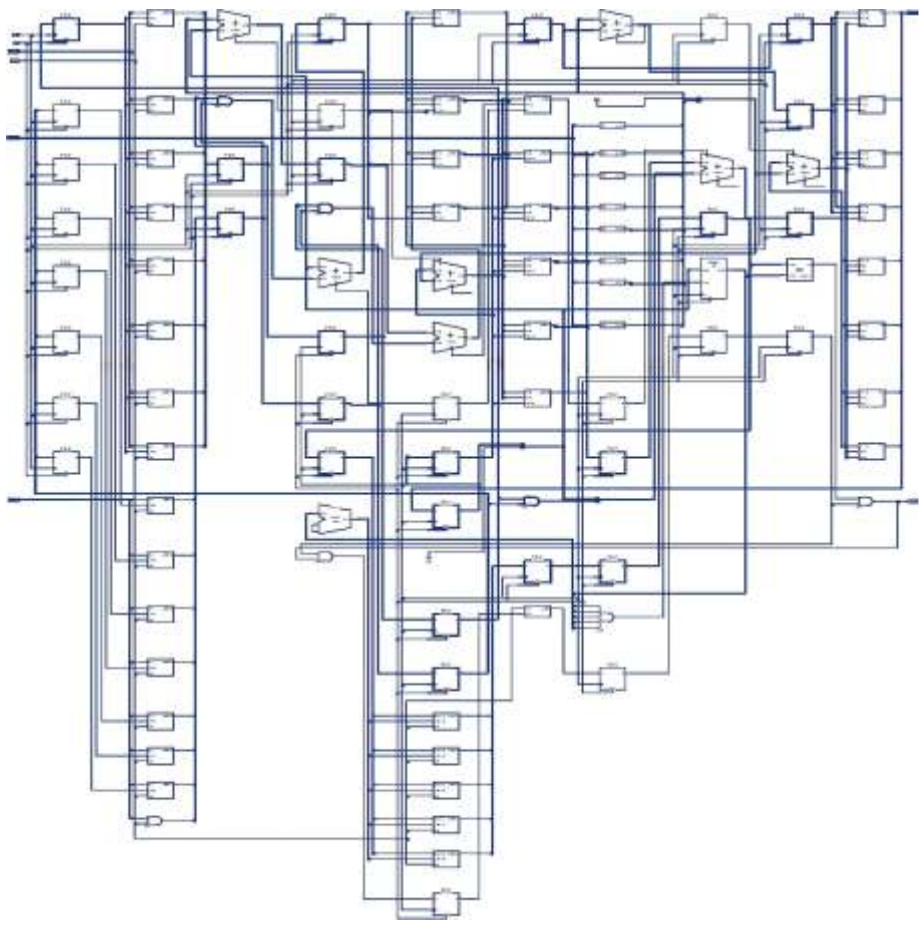

Fig. 13. Pipelined Modular Multiplier Block Diagram.

▪ Macro Statistics

- Adders/Subtractors: 8
- 4-bit adder carry out: 4
- 5-bit adder carry in/out: 3

- 5-bit subtractor: 1
- Counters: 1
- 5-bit up counter: 1
- Registers: 134
- Flip-Flops: 134
- Comparators: 1
- 5-bit comparator equal: 1

Comparisons and analysis remarks

Based on the results obtained from the *160-bit* synthesis, the pipelined Homogenous ECC point addition and the parallel Homogenous ECC point addition have been compared. For pipelined point–addition, the longest path in the implementation will be the path of a modular addition. This is because modular addition operations are given their own designated clock cycles separate than the multiplications. Note that the critical path of the modular multiplication is less than the modular addition. Therefore, the period used for pipelined point addition will depend on the longer path, i.e. the modular addition. The reason that modular addition takes a longer period is that the critical path depends on the full-adder of the carry-save adder along with the $k/2+1$ carry-propagate adder.

In contrast, the critical path of the modular multiplication depends only on 2-to-1 Mux along with a $k/2$ carry-propagate adder. Therefore, the implementation total period is *7.399ns* with total number of clock cycles: $16N+10$, as discussed earlier. Hence, the total time needed for a point addition using *160-bit* operands is

$$T=7.399n*(16*160 + 10)=19.01\textit{microsecond}.$$

For a parallel implementation of the point-addition, the longest period includes a modular multiplication and 2 modular additions. This is because the modular additions are implemented in combinational logic.

The total period is: $6.052n + 2*8.359n = 22.77ns$.

The total number of clock cycles is $4N$. Therefore, the total time needed for a point addition using *160-bit* operands is

$$T=(4*160)* 22.77n= 14.57\textit{microseconds}.$$

Using the implemented multiplication and addition units, we compared the proposed design (Figure 7) with previous parallel design shown in Figure 6, both studied in relation to their area and time. Since the basic components are the same implemented in FPGA, the comparison is believed to be fair and very practical. The study considered the area and timing estimations. To make our study consistent with the previous study in Gutub & Ibrahim (2003), we assume the basic hardware unit as the multiplier. All other units are quantified relative to this multiplier unit, as summarized in Table 3.

Conclusion

In this paper, we redesigned multiplier hardware as pipelined for Elliptic Curve Cryptography (ECC) computations. The design adopted projective coordinates ECC arithmetic to reduce the inversion complexity. The pipelined architecture is implemented and synthesized through a Xilinx Virtex-4 FPGA platform for 160-bits. Based on the synthesis results provided, we concluded that the parallel implementation of the point addition is faster than our pipelined approach; however, the pipelined approach is more advanced in term of chip area. Combing the speed and area as a figure of merit cost-values showed that this work gave overall efficiency in its area time cost which made it very attractive showing a promising research direction for researchers to work on.

Table 3. Cost comparison.

Total Time ***(microsecond)*** *(for 160-bits ECC point addition operation)*		**Pipelined**	**Parallel**	**Percentage**
		19.01	*14.57*	~30% more delay
Cost approximation based on similar Area estimate considered in previous designs [33]	Area (figure relating to size of hardware)	*2*	*4*	~50% efficient area
	AT (Area × Time)	*38*	*58*	~34% efficient cost
	AT^2 (Area × Time × Time)	*722*	*845*	~15% efficient cost
	A^{2}T (Area × Area × Time)	*76*	*232*	~67% efficient cost
Cost approximation based on this work implementation Area estimate	Area (figure relating to size of hardware)	*141+421=562*	*82+4*233=1014*	~45% efficient area
	AT (Area × Time)	*10,678*	*14,804*	~28% efficient cost
	AT^2 (Area × Time × Time)	*202,882*	*216,144*	~6% efficient cost
	A^{2}T (Area × Area × Time)	*6,001,036*	*15,011,256*	~60% efficient cost

ACKNOWLEDGEMENT

We would like to thank King Fahd University of Petroleum and Minerals (KFUPM) for partially hosting this research. Special appreciation to the students of the course COE 509, Applied Cryptosystems - Techniques & Architectures, for their valuable initiatives and positive cooperation. Thanks to both research centers: The Center of Excellence in Information Assurance (CoEIA), King Saud University, Riyadh, and The Center of Research Excellence in Hajj and Omrah, Umm Al-Qura University (UQU), Makkah, for collaborative moral support toward the achievements in this work.

REFERENCES

Akishita, T. 2001. Fast simultaneous scalar multiplication on elliptic curve with Montgomery form. Selected Areas in Cryptography (SAC). LNCS **2259**: 255-267.

Ansari, B. & Hasan, M.A. 2006. High performance architecture of elliptic curve scalar multiplication. Technical Report CACR 2006-01.

Bajracharya, S., Shu, C., Gaj, K. & El-Ghazawi, T. 2004. Implementation of elliptic curve cryptosystems over GF(2n) in optimal normal basis on a reconfigurable computer. Field-Programmable Logic and Applications (FPL), LNCS **3203**: 1001-1005.

Batina, L., Bruin-Muurling, G. & Örs, S.B. 2004. Flexible hardware design for RSA and elliptic curve cryptosystems. The Cryptographer's Track at RSA Conference (CT-RSA), LNCS **2964**:250-263, San Francisco,CA, USA, February 23-27, 2004, Springer_verlag.

Bai, G., Chen, G. & Chen, H. 2005. Fast scalar multiplications of elliptic curve cryptosystems over binary fields. SKLOIS Information Security and Cryptology (CISC): 315-323.

Bajard, J.-C., Imbert, L., Negre, C. & Plantard, T. 2003. Efficient multiplication in GF(pk) for elliptic curve cryptography. IEEE Symposium on Computer Arithmetic (ARITH-**16**): 181-187.

Bednara, M., Daldrup, M., Teich, J., Gathen, J. von zur & Shokrollahi, J. 2002. Tradeoff analysis of FPGA-based elliptic curve cryptography. IEEE Symposium on Circuits and Systems (ISCAS) **5:**797-800.

Bednara, M., Daldrup, M., Gathen, J. von zur, Shokrollahi, J. & Teich, J. 2002. Reconfigurable implementation of elliptic curve crypto algorithms. IEEE Parallel and Distributed Processing Symposium (IPDPS): 157 – 164.

Bertoni, G., Guajardo, J., Kumar, S., Orlando, G., Paar, C. & Wollinger, T., 2003. Efficient GF(pm) arithmetic architectures for cryptographic applications. The Cryptographer’s Track at RSA Conference (CT-RSA), LNCS **2612:** 158- 175. San Francisco, CA. USA, April 2003, Springer_verlag.

Blake, I., Seroussi, G. & Smart, N. 1999. Elliptic Curves in Cryptography. Cambridge University Press, New York.

Certicom Research, 2000. SEC 2: Recommended elliptic curve domain parameters, v1.0. Available online from: <http://www.secg.org/>.

Chen, G., Bai, G. & Chen, H. 2007. A high-performance Elliptic Curve Cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit. IEEE Transactions on

Circuits and Systems II, **54(5):** 412-416.

Cheung, R.C.C., Luk, W. & Cheung, P.Y.K. 2005. Reconfigurable elliptic curve cryptosystems on a chip. Design, Automation and Test in Europe (DATE) **1:**24-29.

Chung, J.W., Sim, S.G. & Lee, P.J. 2000. Fast implementation of elliptic curve defined over GF(pm) on CalmRISC with MAC2424 coprocessor. Cryptographic Hardware and Embedded Systems (CHES), LNCS **1965**: 57-70.

Daneshbeh, A. K. & Hasan, M.A. 2004. Area efficient high speed elliptic curve cryptoprocessor for random curves. IEEE Symposium on Information Technology: Coding and Computing (ITCC) **2:**588-592.

Digital Signature Standard (DSS) 2000. U.S. Department of Commerce/National Institute of Standards and Technology (NIST). FIPS PUB **182-2** change1.

Dyka, Z. & Langendoerfer, P. 2005. Area efficient hardware implementation of elliptic curve cryptography by iteratively applying Karatsuba's method. Design, Automation and Test in Europe (DATE) **3:** 70-75.

Ernst, M., Klupsch, S., Hauck, O. & Huss, S.A. 2001. Rapid prototyping for hardware accelerated elliptic curve public-key cryptosystems. IEEE Rapid System Prototyping (RSP), Pp. 24-31.

Eberle, H., Gura, N. & Chang-Shantz, S. 2003. A cryptographic processor for arbitrary elliptic curves over GF(2m). Application-Specific Systems, Architectures, and Processors (ASAP), Pp. 444-454.

Eberle, H., Gura, N., Shantz, S.C., Gupta, V., Rarick, L. & Sundaram, S. 2004. A public-key cryptographic processor for RSA and ECC. Application-Specific Systems, Architectures, and Processors (ASAP), Pp. 98-110.

Ernst, M., Jung, M., Madlener, F., Huss, S. & Blumel, R. 2002. A reconfigurable system on chip implementation for elliptic curve cryptography over GF(2m). Cryptographic Hardware and Embedded Systems (CHES), LNCS **2523:** 381-399.

Grabbe, C., Bednara, M., Gathen, J. von zur, Shokrollahi, J. & Teich, J. 2003. A high Performance VLIW processor for finite field arithmetic. IEEE Parallel and Distributed Processing Symposium (IPDPS): 189, Nice, France.

Gura, N., Shantz, S.C., Eberle, H., Finchelstein, D., Gupta, S., Gupta, V. & Stebila, D. 2002. An end-to-end systems approach to elliptic curve cryptography. Cryptographic Hardware and Embedded Systems (CHES), LNCS **2523**: 349-365.

Gutub, A. & Ibrahim, M.K. 2003. High Radix Parallel Architecture For GF(p) Elliptic Curve Processor. IEEE Conference on Acoustics, Speech, and Signal Processing (ICASSP) Pp. 625- 628, Hong Kong.

Gutub, A. 2006. Merging GF(p) Elliptic curve point adding and doubling on pipelined VLSI cryptographic ASIC architecture. International Journal of Computer Science and Network Security (IJCSNS) **6(3A):** 44–52.

Gutub, A. 2007. Area Flexible GF(2k) Elliptic curve cryptography coprocessor. International Arab Journal of Information Technology (IAJIT) **4(1):**1-10.

Hauck, O., Katoch, A. & Huss, S.A. 2000. VLSI system design using asynchronous wave pipelines: A 0.35µm CMOS 1.5GHz elliptic curve public key cryptosystem chip. Symposium on Advanced Research in Asynchronous Circuits and Systems (ASYNC), Pp. 188-197.

Huss, S.A., Jung, M. & Madlener, F. 2004. High speed elliptic curve crypto processors: Design space exploration by means of reconfigurable hardware. Proceedings of the 6th International Scientific and Applied Conference - Information Security, Taganrog, Russian Federation, July 2004.

Itoh, K., Takenaka, M., Torii, N., Temma, S. & Kurihara, Y. 1999. Fast implementation of public-key cryptography on a DSP TMS320C6201. Cryptographic Hardware and Embedded Systems (CHES), LNCS **1717**: 61-72.

Ja'rvinen, K., Tommiska, M. & Skytta J. 2004. A scalable architecture for elliptic curve point multiplication. IEEE Field-Programmable Technology (FPT), Pp. 303-306.

Kerins, T., Marnane, W. & Popovici, E. 2004. Design for reuse of elliptic curve cryptosystem processors for FPGAs. Irish Signals and Systems Conference (ISSC), Pp. 577-582.

Kerins, T., Marnane, W.P. & Popovici, E.M. 2005. An FPGA implementation of a flexible secure elliptic curve cryptography processor. Reconfigurable Computing: Architectures and Applications (ARC). Pp. 22-30, February 2005, IADIS Conference, Algarve, Portugal.

Kumar, S. & Wollinger, T. 2006. Optimum digit serial GF(2m) multipliers for curve-based cryptography. IEEE Trans. Computers. **55**(10):1306-1311.

Leong, P. & Leung, I. 2002. A microcoded elliptic curve processor using FPGA technology. IEEE Transactions on VLSI Systems **10(5):** 550-559.

Mekhallalati, M., Ibrahim, M.K. & Ashur, A. 1996. Radix modular multiplication algorithm. Journal of Circuits and Systems, and Computers, **6(5):** 547-567.

Meurice de Dormale, G. & Quisquater, J. 2007. High-speed hardware implementations of Elliptic Curve Cryptography: A survey. Journal of Systems Architecture **53:** 72-84.

Miyaji, A. 1992. Elliptic curves over FP suitable for cryptosystems. Advances in cryptology-AUSCRUPT'92, Australia.

Moller, B. 2001. Algorithms for multi-exponentiation. Selected Areas in Cryptography (SAC), LNCS **2259**: 165-180.

Okada, S., Torii, N., Itoh, K. & Takenaka, M. 2000. Implementation of elliptic curve cryptographic coprocessor over GF(2m) on an FPGA. Cryptographic Hardware and Embedded Systems (CHES), LNCS **1965**: 25-40.

Okeya, K. & Sakurai, K. 2002. Fast multi-scalar multiplication methods on elliptic curves with precomputation strategy using Montgomery trick. Cryptographic Hardware and Embedded Systems (CHES), LNCS **2523**: 564-578.

Potgieter, M.J. & van Dyk, B.J. 2002. Two hardware implementations of the group operations necessary for implementing an elliptic curve cryptosystem over a characteristic two finite field. IEEE Africon Conference in Africa (AFRICON).

Sakiyama, K., Batina, L., Preneel, B. & Verbauwhede, I. 2007. Multicore curve-based cryptoprocessor with reconfigurable modular arithmetic logic units over GF(2n). IEEE Transactions on Computers **56**(9): 1269-1282.

Satoh, A. & Takano, K. 2003. A scalable dual-field elliptic curve cryptographic processor. IEEE Transactions Computers **52**(4): 449-460.

Solinas, J.A. 2001. Low-weight binary representations for pairs of integers. Tech. Report CORR 01-41, Department of Combinatorics & Optimization. Available online from: <http://wwwcacr.math.uwaterloo.ca/techreports/2001/corr2001-41.ps>.

Tawalbeh, L., Mohammad, A. & Gutub, A. 2010. Efficient FPGA implementation of a programmable architecture for GF(p) elliptic curve crypto computations. Journal of Signal Processing Systems, Springer **59**(3): 233-244.

Telle, N., Luk, W. & Cheung, R.C.C. 2004. Customising hardware designs for elliptic curve cryptography. Computer Systems: Architectures, Modeling, and Simulation (SAMOS), Pp. 274-283.

Gathen, J. von zur & Shokrollahi, J. 2005. Efficient FPGA-based karatsuba multipliers for polynomials over F2, in: Selected Areas in Cryptography (SAC), LNCS **3897:** 359-369.

Lutz, J. & Hasan, M.A. 2004. High performance FPGA based elliptic curve cryptographic co-processor. IEEE Symposium on Information Technology: Coding and Computing (ITCC), **2:** 486-492.

McIvor, C., McLoone, M. & McCanny, J. 2004. An FPGA elliptic curve cryptographic accelerator over GF(p). Irish Signals and Systems Conference (ISSC), Pp. 589-594.

Mentens, N., Örs, S.B. & Preneel, B. 2004. An FPGA implementation of an elliptic curve processor GF(2m). 14th ACM Great Lakes Symposium on VLSI (GLSVLSI'04). Pp. 454-457.

Nguyen, N., Gaj, K., Caliga, D. & El-Ghazawi, T., 2003. Implementation of elliptic curve cryptosystems on a reconfigurable computer. IEEE Field-Programmable Technology (FPT), Pp. 60-67.

Orlando, G. & Paar, C. 2001. A scalable GF(p) elliptic curve processor architecture for programmable hardware. Cryptographic Hardware and Embedded Systems (CHES), LNCS **2162**: 356-371.

Orlando, G. & Paar, C. 2000. A high-performance reconfigurable elliptic curve processor for GF(2m). Cryptographic Hardware and Embedded Systems (CHES), LNCS **1965**: 41-56.

Örs, S.B., Batina, L., Preneel, B. & Vandewalle, J., 2003. Hardware implementation of an elliptic curve processor over GF(p). Application-Specific Systems, Architectures, and Processors (ASAP), Pp. 433-443.

Rodríguez-Henríquez, F. & Koç, Ç.K. 2003. On fully parallel karatsuba multipliers for GF(2m). Computer Science and Technology (CST), Pp. 405-410.

Saqib, N.A., Rodríguez-Henríquez, F. & Díaz-Pérez, A. 2004. A parallel architecture for computing scalar multiplication on hessian elliptic curves. Symposium on Information Technology: Coding and Computing (ITCC), **2:**493-497.

Saqib, N.A., Rodríguez-Henríquez, F., & Díaz-Pérez, A., 2004. A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2m). 18th International Parallel & Distributed Processing Symposium (RAW2004), IEEE Computer Society Press, Santa Fe, New Mexico, Pp. 144-154

Shu, C., Gaj, K., & El-Ghazawi, T. 2005. Low-latency elliptic curve cryptography accelerators for NIST curves on binary fields. IEEE Field-Programmable Technology (FPT), Pp. 309-310.

Smart, N.P. 2001. The hessian form of an elliptic curve. Cryptographic Hardware and Embedded Systems (CHES), LNCS **2162:** 118-125.

Sozzani, F., Bertoni, G., Turcato, S., & Breveglieri, L., 2005. A parallelized design for an elliptic curve cryptosystem coprocessor. Symposium on Information Technology: Coding and Computing (ITCC) **1:**626-630.