

Pipelining GF(P) Elliptic Curve Cryptography Computation

Adnan Abdul-Aziz Gutub, Mohammad K. Ibrahim^{*}, and Ahmad Kayali

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran 31261, Saudi Arabia
gutub@kfupm.edu.sa

Abstract

This paper proposes a new method to compute Elliptic Curve Cryptography in Galois Fields GF(p). The method incorporates pipelining to utilize the benefit of both parallel and serial methodology used before. It allows the exploitation of the inherited independency that exists in elliptic curve point addition and doubling operations. The results showed attraction because of its improvement over many parallel and serial techniques of elliptic curve crypto-computations.

1. Introduction

Elliptic Curve Cryptography (ECC) is a public-key crypto-system proposed by Koblitz [15] and Miller [16] in 1985. The idea of ECC is based on the Discrete Logarithm problem over the points on an elliptic curve. Since 1985, the year ECC was introduced, no real breakthroughs have been made in determining security weaknesses in the algorithm [1-9]. Although evaluators are still unconvinced to the trustworthiness of this technique, several cryptographic applications have been developed lately using these properties. The main improvement of ECC when compared to other equal security cryptosystems (e.g. RSA [17]) is found in the significant reduction in its key size [2,5,8], which results in a substantial faster system.

Several GF(p) ECC processors have been proposed in the literature [4,7,10,12]. The gain of using dedicated hardware as crypto-systems is that it results in a considerable speed improvement and power reduction when compared to software solutions on general

purpose programmable processors. It also provides higher security than software solutions [10].

The proposed method considers representing the elliptic curve points as projective coordinate points in order to reduce the number of all inversion operations to one, to enhance the overall performance as adopted in many techniques [4,6,12]. This work, however, differs from existing ones in departing from the current sequential and parallel approaches in the computation of ECC to pipelining in four-stages. It is shown that pipelining will improve the speed and complexity over the sequential and parallel approaches, actually gaining the benefit of both, as will be proven by the AT characteristics.

In the next section, we give an idea of encryption and decryption using ECC. Then, in Section 3, the ECC projective coordinates arithmetic is presented to avoid the complexity of the inverse computations. Section 3 also maps the projective coordinate procedures into dataflow graphs showing data dependencies. Section 4 provides the new pipelined method. The pipelining verification is provided in Section 5. In Section 6, we present the concluding comparisons.

2. Encryption and Decryption

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 compute 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 any one 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

$$C_m = \{k(x, y), (x_m, y_m) + kP_A\}.$$

^{*} Professor Mohammad K. Ibrahim is now with the DE MONTFORT University at the United Kingdom. He can be contacted through email: ibrahim@dmu.ac.uk

The ciphertext pair of points uses A's public key, where only user A can decrypt the plaintext using his private key.

To decrypt the ciphertext C_m , the first point in the pair of $C_m, k(x, y)$, is multiplied by A's private key to get the point: $n_A(k(x, y))$. Then 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 operation is:

$$\begin{aligned} & ((x_m, y_m) + kP_A) - n_A(k(x, y)) = \\ & (x_m, y_m) + k(n_A(x, y)) - n_A(k(x, y)) = \\ & (x_m, y_m) \end{aligned}$$

In fact, the most time consuming operation in the encryption and decryption procedure is finding the multiples of the base point, (x, y) , known as scalar multiplication. Assume the base point is (x, y) is noted as P . The ECC algorithm is to calculate nP from P , which is performed by the binary method, since it is known to be efficient and practical to implement in hardware [2,5,7,9,10]. The binary method for scalar multiplication over an elliptic curve can be easily adapted from the corresponding algorithms used in exponentiation, and are summarized below:

Define n : number of bits in k and k_i : the i^{th} bit of k
 Input: P (a point on the elliptic curve).
 Output: $Q=kP$ (another point on the curve).

Left to Right Algorithm

1. if $k_{k-1} = 1$, then $Q := P$ else $Q := 0$;
2. for $i = k-2$ down to 0 ;
3. { $Q := Q + Q$;
4. if $k_i = 1$ then $Q := Q + P$; }
5. return Q ;

Right to Left Algorithm

1. if $k_{k-1} = 1$, then $Q := P$ else $Q := 0$;
2. for $i = k-2$ down to 0 ;
3. $P := P + P$;
4. if $k_i = 1$ then $Q := Q + P$;
5. return Q ;

The basic operations in all scalar multiplication algorithms are point addition and point doubling over an elliptic curve. Both binary methods algorithm scan the binary bits of k 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 $Q+P$.

The left to right algorithm has received more attention due to its efficiency which aiming to minimize the number of non zero bits in k best for sequential computation. However, the right to left algorithm is more suitable for parallel or pipelined execution since the doubling can be carried out independently of the point addition.

As mentioned earlier, the points operations over elliptic curve requires inversion [10]. This inversion is the most expensive operation [13,18-20] within all ECC calculations. However, there are designs that replace the inversion by several multiplication operations by representing the elliptic curve points as projective coordinates as will be clarified next.

3. Projective Coordinate Arithmetic

The projective coordinates are to eliminate the need for performing inversion. For elliptic curve defined over $GF(p)$, the normal elliptic point (x, y) is projected to (X, Y, Z) , where $x=X/Z$, and $y=Y/Z$ [1]. This transformation computation to projective coordinates is performed only twice: at the beginning and at the end, so they can be calculated in software or by the main processor.

The form procedure for point addition is shown below:

$P = (X_1, Y_1, Z_1); Q = (X_2, Y_2, Z_2); P+Q = (X_3, Y_3, Z_3);$
 where $P \neq \pm Q$

$(x, y) = (X/Z, Y/Z) \rightarrow (X, Y, Z)$

$\lambda_1 = X_1 Z_1$	<i>1M</i>
$\lambda_2 = X_2 Z_1$	<i>1M</i>
$\lambda_3 = \lambda_2 - \lambda_1$	
$\lambda_4 = Y_1 Z_2$	<i>1M</i>
$\lambda_5 = Y_2 Z_1$	<i>1M</i>
$\lambda_6 = \lambda_5 - \lambda_4$	
$\lambda_7 = \lambda_1 + \lambda_2$	
$\lambda_8 = \lambda_6^2 Z_1 Z_2 - \lambda_3^2 \lambda_7$	<i>5M</i>
$Z_3 = Z_1 Z_2 \lambda_3^3$	<i>2M</i>
$X_3 = \lambda_8 \lambda_3$	<i>1M</i>
$\lambda_9 = \lambda_3^2 X_1 Z_2 - \lambda_8$	<i>1M</i>
$Y_3 = \lambda_9 \lambda_6 - \lambda_3^3 Y_1 Z_2$	<i>2M</i>

	<i>15M</i>

Similarly, the form procedure for point doubling is shown below:

$P = (X_1, Y_1, Z_1) ; P+P = (X_3, Y_3, Z_3)$
 $(x, y) = (X/Z, Y/Z) \rightarrow (X, Y, Z)$

$$\begin{aligned}
\lambda_1 &= 3X_1^2 + aZ_1^2 & 3M \\
\lambda_2 &= Y_1Z_1 & 1M \\
\lambda_3 &= X_1Y_1\lambda_2 & 2M \\
\lambda_4 &= \lambda_1^2 - 8\lambda_3 & 1M \\
X_3 &= 2\lambda_4\lambda_2 & 1M \\
Y_3 &= \lambda_1(4\lambda_3 - \lambda_4) - 8(Y_1\lambda_2)^2 & 3M \\
Z_3 &= 8\lambda_2^3 & 2M \\
&----- \\
& & 13M
\end{aligned}$$

The squaring calculation in $GF(p)$ is very similar to the multiplication computation. They both are noted as M (multiplication). The number of multiplication processes for adding two points is found to be $15M$, while the number of operations for doubling a point is found to be only $13M$. The number of addition/subtraction operations within the point addition is *six additions*, while they are *four additions* in the point doubling procedure.

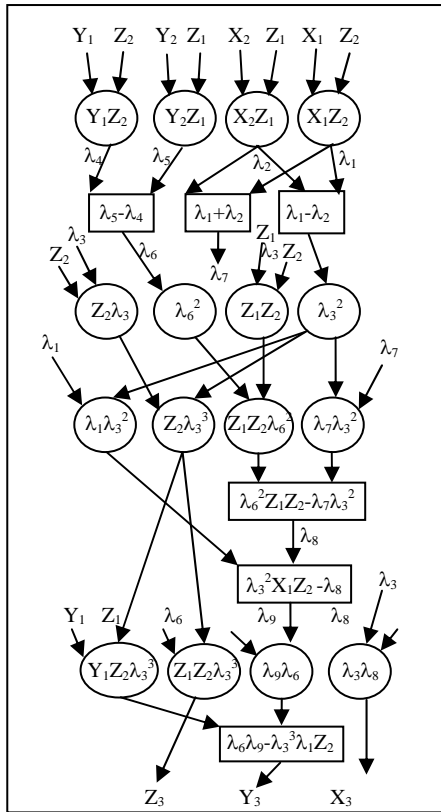


Fig. 1: Dataflow for adding two elliptic curve points

The normal method of ECC computations is to compute the point operations completely sequential, which is too lengthy. Another technique [10] is to try to parallelize the computation on several multipliers as visualized in

the dataflow graphs of Figures 1 and 2, for point addition and doubling respectively. The parallelization is performed depending on the multiplications since their computation complexity makes the addition and subtraction operations negligible. We in this research work are trying to combine between both the parallel and sequential strategies to gain the benefit of both. We improved these ECC methods significantly using the pipelining approach.

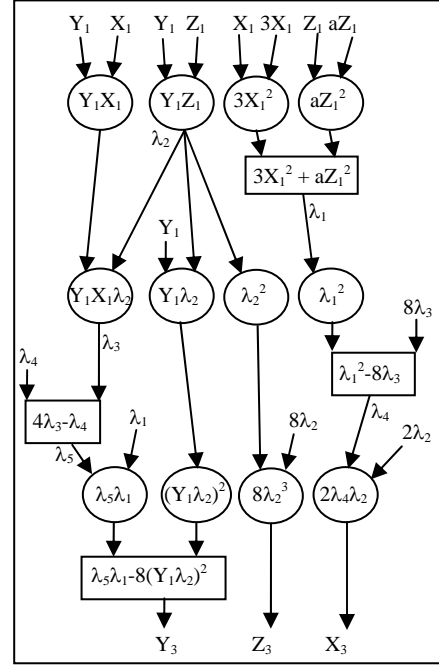


Fig. 2: Dataflow for doubling elliptic curve point

4. Pipelining

Pipelining has the advantage of increasing the throughput, which is the number of results in time unit. However, pipelining will cause additional hardware complexity and time because of the latching between the pipelined stages. The pipelined multiplier assumed in this design consists of four main stages, where each stage should have a well-defined input and output interfaces. Each stage independently processes its inputs and generates the outputs for the next stage. In addition, the addition is made of two stages; i.e. the adder stages are equivalent to two stages of multiplications. Actually, addition is a very fast operation compared to multiplication [13], however, we will assume its computation in two pipelining stages.

The pipeline used for scheduling two points addition operation can be shown in Figure 3. The space

component is the vertical one and time is the horizontal one. We can see that the last stage shows in which register the result will be stored. The total number of registers needed for this pipeline to store intermediate values are seven registers. The total number of time units needed is 31. The operation will store the final results in Registers: R_1, R_3, R_2 . Namely register R_1 will contain the value of Z_3 , R_3 will have X_3 , and R_2 represents Y_3 .

			$Y_1 Z_2 = R_1$	$Y_2 Z_1 = R_2$
		$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$
	$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$
$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	$Z_1 Z_2$
$X_2 Z_1 = R_3$	$X_1 Z_2 = R_4$	$Z_1 Z_2 = R_5$	$R_3 + R_4 = R_2$	
$X_1 Z_2$	$Z_1 Z_2$	$R_3 + R_4$	$R_4 - R_3$	
$Z_1 Z_2$	$R_2 - R_1 = R_1$		$R_1 R_1$	
$R_2 - R_1$		$R_1 R_1$		
$R_4 - R_3 = R_3$	$R_1 R_1 = R_6$		$Z_2 R_3 = R_5$	
$R_1 R_1$		$Z_2 R_3$	$R_6 R_5$	
		$Z_2 R_3$	$R_6 R_5$	$R_3 R_3$
	$Z_2 R_3$	$R_6 R_5$	$R_3 R_3$	
$R_6 R_5 = R_6$	$R_3 R_3 = R_7$			$R_4 R_7 = R_2$
$R_3 R_3$			$R_4 R_7$	$R_2 R_7$
		$R_4 R_7$	$R_2 R_7$	$R_5 R_7$
		$R_4 R_7$	$R_2 R_7$	$R_5 R_7$
$R_2 R_7 = R_4$	$R_5 R_7 = R_5$			$R_2 - R_4 = R_4$
$R_5 R_7$			$R_2 - R_4$	$Z_1 R_5$
		$R_6 - R_4$	$Z_1 R_5$	$Y_1 R_5$
	$R_6 - R_4$	$Z_1 R_5$	$Y_1 R_5$	$R_4 R_3$
$Z_1 R_5 = R_1$	$Y_1 R_5 = R_2$	$R_4 R_3 = R_3$	$R_2 R_1 = R_4$	
$Y_1 R_5$	$R_4 R_3$	$R_2 R_1$		
$R_4 R_3$	$R_2 R_1$			
$R_2 R_1$				$R_4 - R_2 = R_4$

Fig. 3: Two points addition operations pipeline

The pipeline used for scheduling the point doubling operations is shown in Figure 4. The number of registers needed here is six registers, and number of time units is 30. After the completion of the elliptic curve doubling computation the final results are going to be found in registers: R_6, R_3, R_1 , representing the values: Z_3, X_3, Y_3 , respectively.

As can be noticed from Figure 3 and Figure 4, both pipelines are partially utilized, because of the stalls exist. However, we can merge those two pipelines since we can do doubling and addition at the same time. The resulting pipeline is shown in Figure 5.

The total number of registers needed is sixteen registers, and the number of time units is 45. The final results will be found in: $R_1, R_3, R_2, R_{13}, R_{10}, R_8$, which contains the values of: $Z_{3a}, X_{3a}, Y_{3a}, Z_{3b}, X_{3b}, Y_{3b}$, respectively. Note that index a points to result of doubling operation and index b points to the point addition operation.

			$Y_1 X_1 = R_1$	$Y_1 Z_1 = R_2$
		$Y_1 X_1$	$Y_1 Z_1$	$R_3 + X_1$
	$Y_1 X_1$	$Y_1 Z_1$	$X_1 + X_1 = R_3$	$Z_1 a$
$Y_1 X_1$	$Y_1 Z_1$	$X_1 + X_1$	$Z_1 a$	
$R_3 + X_1 = R_3$	$Z_1 a = R_1$		$R_1 R_2 = R_3$	
$Z_1 a$		$R_1 R_2$	$R_3 X_1$	
	$R_1 R_2$	$R_3 X_1$	$R_1 Z_1$	
$R_1 R_2$	$R_3 X_1$	$R_1 Z_1$	$R_2 Y_1$	
$R_3 X_1 = R_1$	$R_1 Z_1 = R_4$	$R_2 Y_1 = R_5$	$R_2 R_2 = R_6$	
$R_1 Z_1$	$R_2 Y_1$	$R_2 R_2$		
$R_2 Y_1$	$R_2 R_2$	$R_3 + R_3 = R_3$	$R_2 + R_2 = R_2$	
$R_2 R_2$	$R_3 + R_3$	$R_2 + R_2$	$R_5 R_5$	
$R_2 + R_3 = R_3$	$R_2 + R_2 = R_4$	$R_5 R_5 = R_4$	$R_4 + R_4 = R_4$	
$R_2 + R_2 = R_2$	$R_5 R_5$	$R_4 + R_4 = R_2$	$R_3 + R_3 = R_2$	
$R_5 R_5$	$R_1 + R_4 = R_1$		$R_1 R_1$	
$R_1 + R_4 = R_2$		$R_1 R_1$	$R_5 + R_5$	
$R_3 + R_3 = R_1$	$R_1 R_1 = R_4$	$R_5 + R_5 = R_5$	$R_4 R_6 = R_6$	
$R_1 R_1$	$R_5 + R_5$	$R_4 R_6$	$R_4 - R_1$	
$R_5 + R_5 = R_5$	$R_4 R_6$		$R_5 + R_5 = R_5$	
$R_4 R_6$				
$R_4 - R_1 = R_1$				$R_1 R_2 = R_3$
$R_5 + R_5 = R_5$		$R_3 - R_1 = R_3$	$R_1 R_2$	
	$R_3 - R_1$	$R_1 R_2$		$R_3 R_1$
	$R_1 R_2$		$R_3 R_1$	
	$R_3 R_1 = R_1$			
$R_3 R_1$				
			$R_1 + R_5 = R_1$	
		$R_1 + R_5$		

Fig. 4: Point doubling operations pipeline

In fact, the pipeline shown in Figure 5 represents the full word length operations. However, we can generalize the pipeline by introducing the size of the digit used in the multiplier:

$$C = 45(N/w)$$

Where, C is the total number of time units
 N is the full word length
 w is the digit size

Therefore, the first four operations in the pipeline: $(Y_1 Z_2), (Y_2 Z_1), (X_2 Z_1)$ and $(X_1 Z_2)$ will be repeated (N/w) times.

			$Y_1 Z_2 = R_1$	$Y_2 Z_1 = R_2$	$X_2 Z_1 = R_3$
		$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$
	$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	$Z_1 Z_2$
$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	$Z_1 Z_2$	$R_2 - R_1$
$X_1 Z_2 = R_4$	$Z_1 Z_2 = R_5$	$R_3 + R_4 = R_2$	$R_4 - R_3 = R_3$		
$Z_1 Z_2$	$R_3 + R_4$	$R_4 - R_3$	$R_1 R_1$		
$R_2 - R_1 = R_1$		$R_1 R_1$	$Y_1 X_1$		
	$R_1 R_1$	$Y_1 X_1$	$Y_1 Z_1$		
$R_1 R_1 = R_6$	$Y_1 X_1 = R_8$	$Y_1 Z_1 = R_9$	$R_{10} + X_1 = R_3$		
$Y_1 X_1$	$Y_1 Z_1$	$R_{10} + X_1$	$Z_1 a$		
$Y_1 Z_1$	$X_1 + X_1 = R_{10}$	$Z_1 a$	$Z_2 R_3$		
$X_1 + X_1$	$Z_1 a$	$Z_2 R_3$	$R_6 R_5$		
$Z_1 a = R_8$	$Z_2 R_3 = R_5$	$R_6 R_5 = R_6$	$R_3 R_3 = R_7$		
$Z_2 R_3$	$R_6 R_5$	$R_3 R_3$	$R_8 R_9$		
$R_6 R_5$	$R_3 R_3$	$R_8 R_9$	$R_{10} X_1$		
$R_3 R_3$	$R_8 R_9$	$R_{10} X_1$	$R_8 Z_1$		
$R_8 R_9 = R_{10}$	$R_{10} X_1 = R_8$	$R_8 Z_1 = R_{11}$	$R_9 Y_1 = R_{12}$		
$R_{10} X_1$	$R_8 Z_1$	$R_9 Y_1$	$R_9 R_9$		
$R_8 Z_1$	$R_9 Y_1$	$R_9 R_9$	$R_{10} + R_{10} = R_{10}$		
$R_9 Y_1$	$R_9 R_9$	$R_{10} + R_{10}$	$R_9 + R_9$		
$R_9 R_9 = R_{13}$	$R_{10} + R_{10} = R_{10}$	$R_9 + R_9 = R_{11}$	$R_{12} R_{12} = R_{12}$		
$R_{10} + R_{10}$	$R_9 + R_9$	$R_{12} + R_{12}$	$R_{11} + R_{11}$		
$R_9 + R_9 = R_9$	$R_{12} + R_{12}$	$R_8 + R_{11} = R_8$	$R_4 R_7$		
$R_{12} + R_{12}$	$R_8 + R_{11}$	$R_4 R_7$	$R_2 R_7$		
$R_{11} + R_{11} = R_{11}$	$R_4 R_7 = R_2$	$R_2 R_7 = R_4$			
$R_4 R_7$	$R_2 R_7$	$R_5 R_7$			
$R_2 R_7$	$R_5 R_7$	$R_8 R_8$			
$R_5 R_7$	$R_8 R_8$	$R_{12} + R_{12}$			
$R_5 R_7 = R_5$	$R_8 R_8 = R_{11}$	$R_{12} + R_{12} = R_{12}$	$R_4 R_3 = R_3$		
$R_8 R_8$	$R_{12} + R_{12}$	$R_{11} R_{13}$	$R_2 R_1$		
$R_{12} + R_{12}$	$R_{11} R_{13}$	$R_6 - R_4$	$R_8 R_9$		
$R_{11} R_{13}$	$R_6 - R_4$				
$R_{11} R_{13} = R_{13}$	$R_{11} - R_8 = R_8$		$R_{10} - R_8 = R_{10}$		
$R_{11} - R_8$	$R_{12} + R_{12}$	$R_{10} - R_8$	$Z_1 R_5$		
$R_{12} + R_{12}$	$R_2 - R_4$	$Z_1 R_5$	$Y_1 R_5$		
$R_2 - R_4$	$Z_1 R_5$	$Y_1 R_5$	$R_4 R_3$		
$Z_1 R_5 = R_1$	$Y_1 R_5 = R_2$	$R_2 R_1 = R_4$	$R_8 R_9 = R_{10}$		
$Y_1 R_5$	$R_4 R_3$	$R_8 R_9$	$R_4 - R_2$		
$R_4 R_3$	$R_2 R_1$		$R_{10} R_8$		
$R_2 R_1$	$R_8 R_9$	$R_{10} R_8$			
$R_4 - R_2 = R_2$	$R_{10} R_8 = R_8$				
$R_{10} R_8$					
			$R_8 - R_{12} = R_8$		
		$R_8 - R_{12}$			

Fig.5: Pipeline for both addition and doubling

5. Pipelining Verification

Pipelining is based on having independent operations that can be done in the same time. To deduce a pipeline for the set of operations needed in the ECC point operations, we need to prove it through the data flow of the operations and check the dependencies. If we take the first part of the data flow, we can see that there are four independent operations that can be achieved simultaneously. Because of the independence of those operations, we can place them in the pipeline and they are noted by four different colors as shown in Figure 6.

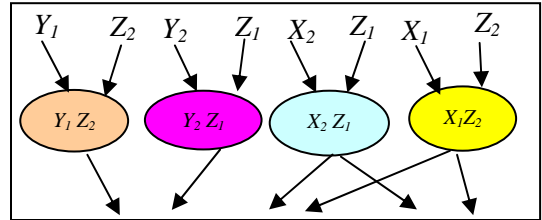


Fig. 6: Four colors of independent operations

Therefore, in the first clock cycle, we place $(Y_1 Z_2)$ into the pipeline. In the next cycle, we move $(Y_1 Z_2)$ to next stage and bring $(Y_2 Z_1)$ into the first stage of the pipeline, and so on. The pipelined version of this portion of the dataflow can be represented as shown in Figure 7.

			$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$
		$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	zzz
	$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	zzz	zzz
$Y_1 Z_2$	$Y_2 Z_1$	$X_2 Z_1$	$X_1 Z_2$	zzz	zzz	zzz

Fig. 7: Four multiplications pipeline dataflow

In each time unit, there are no two slots that have the same color. This shows that those two operations can be achieved simultaneously.

6. Comparisons and Conclusions

As mentioned earlier, the pipelined multiplier needs more area and time than the non-pipelined multiplier used in parallel method. However, we compared the three possible methods: sequential, parallelized and pipelined computations. Tables 1 and 2 compare the estimation of hardware area and time for the three methods. For the sequential scheme, we will need one adder and one multiplier. The parallelized way needs three adders and eight multipliers. Finally, our pipelined technique needs only one multiplier and one adder.

Design	Add's Area	Mult's Area	Total Area
Sequential	24N	71N+71	95N+71
Parallelized	3(24N)	8(71N+71)	640N+568
Pipelined	36N+8	143N+119	179N+127

Table 1: Hardware Component for the three designs

Design	Add's Time	Mult's Time	Total Time
Sequential	10(4N+6)	28(4N ² +16N+16)	112N ² +488N+508
Parallelized	4(4N+6)	4(4N ² +16N+16)	16N ² +80N+88
Pipelined	---	(45/4)(4N ² +28N+24)	45N ² +315N+270

Table 2: Area Component for the three designs

Table 3 shows the AT characteristics for the designs. It is clear from this table (Table 3) that the proposed pipelined computation method beats both the sequential and parallel ways in terms of AT. For the pipelined one, the AT is almost 75% and 78% for the sequential and parallelized method, respectively, for high values of N as observed in Figure 8, which shows the relation between the AT and the number of bits N .

Design	AT
Sequential	10640N ³ +50512N ² +80068N+36068
Parallelized	10240N ³ +60288N ² +101760N+49984
Pipelined	8055N ³ +62100N ² +88335N+34290

Table 3: AT characteristics for the three designs

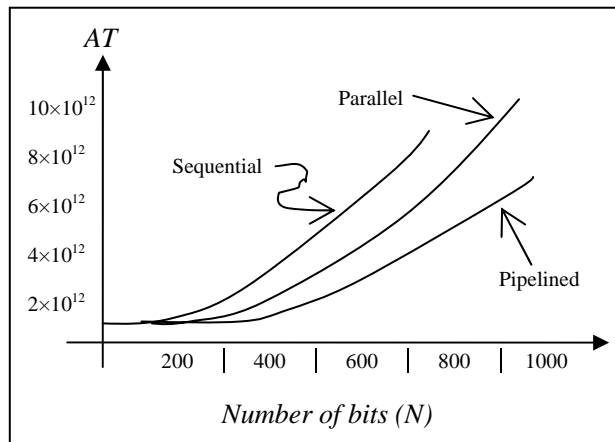


Fig.8: AT study

Acknowledgment

The authors would like to thank the Computer Engineering Department within King Fahd University of Petroleum and Minerals (KFUPM), in Dhahran-Saudi Arabia, for supporting this work.

References

- [1] Miyaji A., "Elliptic Curves over F_p Suitable for Cryptosystems", Advances in cryptology-AUSCRUPT'92, Australia, December 1992.
- [2] Stallings, W. "Cryptography and Network Security: Principles and Practice", Second Edition, Prentice Hall Inc., New Jersey, 1999.
- [3] Chung, J., Sim, S., and Lee, P., "Fast Implementation of Elliptic Curve Defined over $GF(p^m)$ on CalmRISC with MAC2424 Coprocessor", Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.
- [4] Okada, S., Torii, N., Itoh, K., and Takenaka, M., "Implementation of Elliptic Curve Cryptographic Coprocessor over $GF(2^m)$ on an FPGA", Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.
- [5] Crutchley, D. A., "Cryptography And Elliptic Curves", Master Thesis under Supervision of Prof. Gareth Jones, submitted to the Faculty of Mathematics at University of Southampton, England, May 1999.
- [6] Orlando, G., and Paar, C., "A High-Performance Reconfigurable Elliptic Curve Processor for $GF(2^m)$ ", Workshop on Cryptographic Hardware and Embedded Systems, CHES 2000, Massachusetts, August 2000.
- [7] Stinson, D. R., "Cryptography: Theory and Practice", CRC Press, Boca Raton, Florida, 1995.
- [8] Paar, C., Fleischmann, P. and Soria-Rodriguez, P., "Fast Arithmetic for Public-Key Algorithms in Galois Fields with Composite Exponents", IEEE Transactions on Computers, Vol. 48, No. 10, October 1999.
- [9] Blake, I., Seroussi, G., and Smart, N., "Elliptic Curves in Cryptography", Cambridge University Press: New York, 1999.

- [10] Gutub, Adnan Abdul-Aziz and Ibrahim, Mohammad K. "High Radix Parallel Architecture For GF(P) Elliptic Curve Processor". IEEE Conference on Acoustics, Speech, and Signal Processing, ICASSP 2003, pages 625- 628, Hong Kong, April 6-10, 2003.
- [11] Mekhallalati, M., Ibrahim, M. K. and Ashur, A, "Radix Modular Multiplication Algorithm", Journal of Circuits and Systems, and Computers, Vol.6, N0.5, pp547-567, 1996
- [12] Orlando, G., and Paar, C., "A scalable GF(p) elliptic curve processor architecture for programmable hardware", Cryptographic Hardware and Embedded Systems, CHES 2001, May 14-15, 2001, Paris, France.
- [13] Crutchley, D. A., "*Cryptography and Elliptic Curves*", Master Thesis under Supervision of Prof. Gareth Jones, submitted to the Faculty of Mathematics at University of Southampton, England, May 1999.
- [14] Piestrak S. J., "Design of High-Speed Residue-to-Binary Number System Converter Based on Chinese Remainder Theorem," *Proceedings of the Int'l Conf. Computer Design (ICCD '94)*, pp. 508-511, Oct. 1994.
- [15] Koblitz, N., "Elliptic Curve Cryptosystems", *Math. Computing*, 1987, 48, pp. 203–209.
- [16] Miller, V., "Use of Elliptic Curves in Cryptography", *Proceedings of Advances in Cryptology (Crypto)*, 1986, pp. 417-426.
- [17] Rivest, Shamir, and Adleman, "A Method for Obtaining Digital Signature and Public-Key Cryptosystems", *Comm. ACM*, Feb. 1978, 21(2), pp. 120-126.
- [18] Gutub, A. and Tenca A., "Efficient Scalable VLSI Architecture for Montgomery Inversion in GF(p)", *Integration, the VLSI Journal*, May 2004, 37(2).
- [19] Gutub, A., Tenca, A., and Koc,C., "Scalable VLSI architecture for GF(p) Montgomery modular inverse computation", *IEEE Computer Society Annual Symposium on VLSI*, April 25-26, 2002.
- [20] Gutub, A., Tenca, A., and Koc,C., "Scalable and Unified Hardware to Compute Montgomery Inverse in GF(p) and GF(2^n)", *Cryptographic Hardware and Embedded Systems - CHES 2002*, pp 485-500, August 13-15, 2002.