HARDWARE REALIZATION OF HIGH SPEED ELLIPTIC CURVE POINT MULTIPLICATION USING PRECOMPUTATION OVER GF(p)

N. Shylashree and V. Sridhar

Department of Electronics and Communication Engg, PESCE, Mandya, India

ABSTRACT

Two new theoretical approaches for the hardware realization of high speed elliptic curve point multiplication over a prime field (GF(p)) are presented. These hardware implementations use multiple units of elliptic curve point doublers, point adders and multiplexers. The modular hardware approach used here provides high speed and scalability.

Keywords: Elliptic Curve Point Adders, Point Doublers, Elliptic Curve Point Multiplier, Point Multiplexers, Galois Field

1. INTRODUCTION

In Diffi-Hellman type key agreement (Idrissi et al., 2012), Elliptic Curve Digital Signature Algorithm (Shivkumar and Umamaheswari, 2014) and Elgamal crypto systems (Jie and Kamarulhaili, 2011; Ismail and Hijazi, 2012), we use elliptic curve Point Multiplication (PM) (Hankerson et al., 2004). Several hardware solutions are already available for elliptic curve Point Multiplication. (Ghosh et al., 2007; Orlando and Paar, 2001; De Dormole and Quisquater, 2007).

Our objective is to generate the scalar product kP where P is a point on an elliptic curve over a prime field Fp and k is an integer that belongs to Zp. We propose a fast hardware solution to PM which makes use of hardware Point Doubler (PD) and Point Adder (PA) modules. We describe two different schemes for fast multiplication. In the first method we realize the design for a ‘t’ bit k. Then we extend the design for binary multiples of ‘t’. In the second method multi scalar multiplication is used and the desired result is selected using appropriate multiplexers.

2. BASIC SYMBOLS AND NOTATIONS

Let the given scalar multiplier k be represented in binary as Equation (1):

\[ k = [k_{t-1}, k_{t-2}, \ldots, k_1, k_0] \]  

Here, t is the number of bits of k. That is the size of k when stored in binary is t bits. In terms of these bits k is given by:

\[ k = 2^{t-1}k_{t-1} + 2^{t-2}k_{t-2} + \ldots + 2k_2 + 2k_1 + k_0 \]  

In the light of Equation (2), the product kP can be expressed as Equation (3):

\[ kP = (2^{t-1}k_{t-1}P + 2^{t-2}k_{t-2}P + \ldots + 2^2k_2P + 2k_1P + k_0P) \]  

That is Equation (4 and 5):

\[ kp = \sum_{i=0}^{t-1} 2^iP_k = \sum_{i=0}^{t-1} B_ik_i \]  

Where:

\[ B_i = 2^iP \text{ for } i = 0, 1, 2, \ldots, t-1 \]
2.1. Realization of $B_i k_i$ (First Method)

Consider the term $B_i k_i$. The bit $k_i$ can be either zero or 1. Therefore, multiplication by $k_i$ can be represented as Equation (6):

$$
B_i k_i = 0 \text{ when } k_i = 0 \\
B_i k_i = B_i \text{ when } k_i = 1
$$

(6)

From Equation (5) $B_i k_i$ is equivalent to the logical AND operation as Equation (7):

$$
B_i k_i = B_i \text{ AND } k_i
$$

(7)

The elliptic curve point $B_i$ belonging to the prime field $F_p$ has two components as:

$$
B_i = (x_i, y_i)
$$

where the size of each is $m$ bits. $m$ is given by Equation (8):

$$
m = \left\lfloor \log_2 p \right\rfloor
$$

(8)

Thus the size of $B_i$ is $2m$.

In the hardware realization, $(B_i \text{ AND } k_i)$ can be realized using an array of $2m$ AND-gates. We can also make use of a $2m$ input Controlled Buffer (CB) with an enable control input $EB$ as shown in Fig. 1. When $EB = 0$, the output is zero ($2m$ bits) and when $EB = 1$, output = $B_i = 2P$. Therefore, the Controlled Buffer (CB) realizes Equation (6) are shown in Fig. 1.

2.2. Realization of $kP$

From Equation (3) for $kP$, we can see that $kP$ is obtained as a series of Point Additions. Our aim is to get $kP$ using several $2$ input Point Adders. To realize this, Equation (3) for $kP$ is written as:

$$
kP = Pk_0 + 2Pk_1 + 4Pk_2 + \ldots + 2^{t-2}Pk_{t-2} + 2^{t-1}P
$$

(9)

The RHS of this Equation (9) is grouped as follows:

$$
kP = \left( (Pk_0 + 2Pk_1) + 4Pk_2 + \ldots + 2^{t-1}P \right)
$$

(10)

Then, $kP$ can be obtained as the cumulative sum of 2-input Point Adders. To get that, let us introduce the symbols $Q_1$, $Q_2$, $Q_3$, $Q_4$, $Q_5$, $Q_6$, $Q_7$, $Q_8$, $Q_9$, $Q_{10}$ as follows Equation (11-13):

$$
Q_1 = Pk_0 + 2Pk_1
$$

(10)

$$
Q_2 = Q_1 + 4Pk_2
$$

(11)

$$
Q_3 = Q_2 + 8Pk_3
$$

(12)

$$
Q_{i+1} = Q_i + 2^{i+1}P
$$

(13)

That is Equation (14):

$$
Q_{i+1} = Q_i + 2^{i+1}P
$$

(14)

for $i = 1, 2, \ldots, t-1$.

From Equation (10, 11) substituting for $Q_1$:

$$
Q_2 = Pk_0 + 2Pk_1 + 4Pk_2
$$

(15)

Similarly, from Equation (12) and (15):

$$
Q_3 = Pk_0 + 2Pk_1 + 4Pk_2 + 8Pk_3
$$

(16)

In this way, we can see that:

$$
Q_{t-1} = Pk_0 + 2Pk_1 + 4Pk_2 + \ldots + 2^{t-1}P
$$

(17)

The RHS of Equation (17) is same as $kP$ as given by Equation (3) Thus $kP$ is realized as the Point Sum of $Q_{t-1}$ and $2^{t-1}P$. That is Equation (18):

$$
kP = Q_{t-1} + 2^{t-1}P
$$

(18)

3. HARDWARE REALIZATION FOR AN ‘t’-BIT ‘k’

The elliptical curve Point Multiplier is realized as shown in Fig. 2. Output $kP$ is obtained as the Point Sum of the last Point Adder in a chain of $(t-1)$ Point Adders. In Fig. 2, $t$ cascaded Point Doublers (PD’s) are used to generate $2P$, $4P$, $2^{t-1}P$, $2^tP$. The Controlled Buffers are denoted by CB in Fig. 2. They generate $2^iP$ for $i = 0$ to $(t-1)$. The bit $k_i$ of $K$ and $2^iP$ are the inputs to the corresponding CB. The output of each CB is one of the inputs to the corresponding Point Adder (PA). Equation (10) is realized by Point Adder $PA_i$. Similarly $PA_i$ realizes $Q_i$ as in Equation (11). The last Point Adder $PA_{t-1}$ realizes Equation (13) to give out $Q_{t-1}$ which is the desired output $kP$ itself. The Point Multiplication Module (PMM) provides an additional output $2^tP$ from the last PD block. This output $2^tP$ is used for cascading purpose which will be described later.

The Point Doubling and Addition can be accomplished internally in either affine or projective co-ordinates. In the PMM described in Fig. 2, if say bit $k_i = 0$, we cannot avoid Point Adder $PA_i$ because, next
time, $k_i$ may not be zero. The number of Point Adders is fixed at $(t-1)$ to take care of all possible value of $k$. Therefore, the use of Non Adjacent Form (NAF) representation of $k$ has no benefit in this scheme.

**Fig. 1.** Controlled buffer block

**Fig. 2.** Multiple PD-PA Point Multiplication Module (PMM)
3.1. Timing Analysis of the Proposed PMM

The running time of the PMM shown in Fig. 2 is determined in terms of the running times of Point doublers and Point Adders. All PD’s are similar in structure and working and so also all PA’s are similar. Let D be the time (in an appropriate unit) required by a PD to complete the doubling action and let A be the time required by a PA for Addition. D and A depend on the internal design of the PD’s and PA’s respectively (Hankerson et al., 2004; Ding et al., 2013).

3.1.1. Precomputation

Here, P, 2P,...,2P are precomputed and readily available at the corresponding locations. Now, we need not consider the time taken by PD’s. Consider the time required to get the output Q₁ from PA₁ after applying the input k. Here, inputs k₀, k₁, k₂,...,k₁ are applied simultaneously from a single register holding k. Time taken for signals to pass through CB’s are neglected compared to the time needed at PA’s. Initially, P, 2P, k₀ and k₁ are available at say T₀. Neglecting the time taken by CB’s, Pk₀ and 2Pk₁ are available at the input of PA₁ at T₀ itself. Therefore, the output Q₁ will be ready at T₀+A where A is the time required to generate the output by PA₁. Thus the transition delay at PA₁ is A units of time. After Q₁ is ready, time required by PA₂ to process Q₁ and 4Pk₂ to get Q₂ would be again A. Therefore the total time from T₀ up to the time of getting Q₂ would be A+A=2A. Thus each PA along the chain adds a delay of A and the total delay would be (t-1) A to get kP. Observe that there are (t-1) Point Adders in the additive chain. Therefore the total running time Tᵢ is given by Equation (19):

\[ Tᵢ = (t - 1) A \]  

(19)

3.2. Point Multiplication Module

The Point Multiplication hardware using multiple PD’s and PA’s can be represented by a modular block as shown in Fig. 3. The module is called as PMMᵢ which stands for Point Multiplier Module that gives kP where ‘t’ is the size of ‘k’ in bits. Thus PMM₀ means, the size of ‘k’ is 8 bits. P is the given elliptic curve point of total size 2m.

4. POINT MULTIPLIER MODULES IN CASCADE

When ‘t’ is large, the number of PD’s and PA’s in PMM would also be large. The design and construction of such a large sized Point Multiplier Module becomes practically difficult and can be cumbersome. Therefore, when ‘t’ is large, several smaller sized Point Multiplier Modules are cascaded to realize kP as follows.

Let the smaller size chosen be w bits. The binary representation of ‘k’ is partitioned into ‘d’ words of size ‘w’ bits each. The value of ‘d’ is given by Equation (20):

\[ d = \left\lceil \frac{t}{w} \right\rceil \]  

(20)

If ‘t’ is not perfectly divisible by ‘w’, binary representation of ‘k’ is padded with d*n-t zeros on the left hand side (De Dormole and Quisquater, 2007). The partition of ‘k’ into ‘d’ words is shown in Fig. 4. Let K₀, K₁,..., Kᵢ₋₁ be the decomposed binary words of ‘k’. Now ‘k’ can be expressed in terms of K₀, K₁,..., Kᵢ₋₁, Kᵢ in base 2 as Equation (21) (Shivkumar and Umamaheswari, 2014):

\[ k = [K_d, ..., K_1, K_0]_{2w} \]  

(21)

The numerical value of ‘k’ in terms of K₀,..., Kᵢ₋₁, Kᵢ can be expressed as:

\[ k = 2^{(d-1)}K_d, +, ..., +2^{w}K_w, + 2^{w}K_w, + K_0 \]  

(22)

Now, in the light of Equation (22), the product kP can be written as Equation (23):

\[ kP = 2^{(d-1)}PK_d, +, ..., +2^{w}PK_w, + 2^{w}PK_w, + PK_0 \]  

(23)

That is Equation (24 and 25):

\[ kP = \sum_{i=0}^{d-1} P \times K_i \]  

(24)

Where:

\[ P_i = 2^{w}P \text{ for } 0 \leq i \leq (d - 1) \]  

(25)

K₀, K₁,..., Kᵢ₋₁ are of size w bits each and the RHS of Equation (23) has d terms. Therefore, d number of cascaded PMMᵢ’s can realize Equation (23) to get kP as shown in Fig. 5. Equation (23) can be expressed in terms of partial sums S₁, S₂, ..., Sᵢ₋₁ as follows Equation (26-28):

Let \[ S_j = 2^{w}PK_j + PK_0 = P \times K_j + P \times K_0 \]  

(26)

Then, \[ S_j = 2^{w}PK_j + S_i = P \times K_j + S_i = 2^{w}PK_j + 2^{w}PK_j + PK_0 \]  

(27)
Fig. 3. Point Multiplier Module for a ‘t’ bit ‘k’

\[ k = \left[ k_{d-1} \ldots k_{(d-1)w} \right| \ldots \left| k_{2w-1} \ldots k_{w+1} k_w \right| k_{w-1} \ldots k_0 ] \]

\[ K_{d-1} \quad K_1 \quad K_0 \]

Fig. 4. Partition of k into d words of w bits each

Fig. 5. Cascaded PMM’s for large K
\[ S_{d-1} = 2^{d-1}PK_{d-1} + S_{d-2} = P_{d-1}K_{d-1} + S_{d-2} \]  
\[ = 2^{d-1}PK_{d-1} + ... + 2^0PK_1 + 2^0PK_0 + PK_o \]  

From Equation (28, 23) we see that Equation (29):

\[ kP = S_{d-1} \]  

Realization of these partial sums is shown in Fig. 5. Pi’s are realized as \( P_i = 2^iP_{i-1} \) for \( 0 \leq i \leq (d-1) \) with \( P_0 = P \). The output of Point Adder \( PA_1 \) is \( S_1 \). The inputs to get \( S_1 \) are \( P_1K_1 \) and \( S_0 \) and so on. The output of the last Point Adder gives \( S_{d-1} \) which is same as \( kP \). Additional output \( 2^wP \) can be used for further cascading.

4.1 Total Number of PD’s and PA’s in Cascaded PMM

Each PMMW uses \((w-1)\) number of PA’s and ‘\( w \)’ number of PD’s. There are ‘\( d \)’ number of PMMW’s and \((d-1)\) number of PA’s in Fig. 5. Therefore the total number of PD’s is \( dw \) which is equal to ‘\( t \)’. The number of PA’s is \( d(w-1) + (d-1) = dw-1 = t-1 \). Since ‘\( t \)’ the size of ‘\( k \)’ can go up to \( m \) (size of \( p \)), the number of PD’s is \( m \) and the number of PA’s is \((m-1)\).

4.2 Timing Analysis of the Cascaded PMM

The timing analysis of the cascaded PMM’s is determined with precomputation of \( 2^wP \) for \( w=1,2,\ldots\) etc.

4.2.1 Precomputation

All inputs \( K_0, K_1,\ldots, K_{d-1} \) are applied simultaneously. Consider the inputs \( K_0P_0 \) and \( K_1P_1 \) to \( PA_1 \) in Fig. 5. The delay due to \( PMMW_1 \), the \( PMMW \), identified by 1 in Fig. 5, for signal \( K_0P_0 \) is \( (w-1)A \) as given by Equation (19).

Thus equation (30) and (31) becomes:

\[ \text{delay}(K_0P_0) = (w-1)A \]  
\[ \text{delay}(K_0P_0) = (w-1)A \]  

Therefore, both of them are available after a delay of \( (w-1)A \) at the input of \( PA_1 \). Therefore the delay of \( S_1 \) is, delay \( (S_1) = \) of Equation (32):

\[ (w-1)A + A = wA \]  

Now, consider the inputs to \( PA_2 \) which are \( S_1 \) and \( K_2P_2 \). Delay of \( K_2P_2 \) due to \( PMMW \) (2) is Equation (33):

\[ \text{delay}(K_2P_2) = (w-1)A \]  

From Equation (32 and 33), both \( S_1 \) and \( K_2P_2 \) are available at the input of \( PA_2 \) after a delay of \( wA \). To this, adding the delay in \( PA_2 \), we get Equation (34):

\[ \text{delay}(S_2) = wA + A = (w+1)A \]  

In this way, each PA in the adder chain adds a delay of \( A \). Thus \((d-1)\) PA’s add a delay of \((d-1)A \). Initial delay at the input of \( PA_1 \) is \((w-1)A \). Hence the total delay of \( S_{d-1} \) is:

\[ \text{delay}(S_{d-1}) = (w-1)A + (d-1)A \]  

Therefore the total delay of signal \( kP \) is Equation (35):

\[ \text{delay}(kP) = (w + d - 2)A \]  

4.3 Register Size Requirement for \( PMMW \)

In \( PMMW \), let the size of \( P \) be \( N\)-bits Equation (36-38):

\[ \text{the size of } 2^0P = N \]  
\[ \text{the size of } 2^wP = N + w \]  

Here \( N \) is the NIST standard Value for ECC. Thus, the sizes will be increasing progressively for each succeeding stage and this should be taken care off during the realization of the modules. However, except for the register sizes the modules are similar.

5. BASIC PRINCIPLE FOR AN 8-BIT ‘\( k \)’ WITHOUT CB (SECOND METHOD)

Let \( P \) be a given point on the elliptic curve \( E \) (\( F_p \)). Let ‘\( k \)’ be an 8 bit integer belonging to \( Z_p \). The objective is to generate \( kP \) as fast as possible.

5.1 Precomputation

Assuming that \( P \) is known in advance, we precompute \( 2P, 4P,\ldots,128P \) using the Point Doubleers. We also precompute the following additive terms using Point Adders as shown in Fig. 6. 3 \( P \) using \( P + 2P \), 12P using \( 4P + 8P, 48P \) using \( 16P + 32P \) and 192P using \( 64P + 128P \).
In Fig. 6, PD is a Point Doubler and PA is a Point Adder. The hardware precomputation module uses 8 PD’s and 4 PA’s. These precomputed values are used later in the realization of the fast multiplier. The last PD in Fig. 6 generates 256P. This value is needed for further concatenation which will be described later.

5.2. Expression for kP

The 8 bit integer k is written in binary as Equation (39):

\[ k = \left[ k_7, k_6, k_5, k_4, k_3, k_2, k_1, k_0 \right] \]  

The decimal value of k in terms of its binary digits is Equation (40):
\[ k = 128k_x + 64k_y + 32k_z + 16k_d + 8k_x + 4k_z + 2k + k_0 \]  

Therefore \( kp \) is:

\[ kp = 128pk_x + 64pk_y + 32pk_z + 16pk_d + 8pk_x + 4pk_z + 2pk + pk_0 \]  

The RHS of the above equation is formatted into 4 sub groups as:

\[ \left( 128pk_x + 64pk_y \right) + \left( 32pk_z + 16pk_d \right) \]

\[ + \left( 8pk_x + 4pk_z \right) + \left( 2pk + pk_0 \right) \]  

The sub groups are represented by \( Q_0, Q_1, Q_2 \) and \( Q_3 \) as:

\[ Q_0 = \left( 128pk_x + 64pk_y \right) \]  

\[ Q_1 = \left( 32pk_z + 16pk_d \right) \]  

\[ Q_2 = \left( 8pk_x + 4pk_z \right) \]  

\[ Q_3 = \left( 2pk + pk_0 \right) \]  

Then in the light of Equation (42-45), Equation (41) becomes Equation (46):

\[ kp = Q_0 + Q_1 + Q_2 + Q_3 \]  

This can be rewritten as Equation (47):

\[ kp = (Q_0 + Q_1) + (Q_2 + Q_3) \]  

Now let us consider \( Q_0 \) as given by Equation (45):

When \( k_0 = 0 \) and \( k_1 = 0 \), \( Q_0 = 0 \)

When \( k_0 = 1 \) and \( k_1 = 0 \), \( Q_0 = P \)

When \( k_0 = 0 \) and \( k_1 = 1 \), \( Q_0 = 2P \)

When \( k_0 = 1 \) and \( k_1 = 1 \), \( Q_0 = 2P + P = 3P \)

This can be written in a tabular form as shown in Table 1.

| \( k_1 \) | \( k_0 \) | \( Q_0 \) |
|---------|---------|---------|
| 0       | 0       | 0       |
| 0       | 1       | P       |
| 1       | 0       | 2P      |
| 1       | 1       | 3P      |

### 6. HARDWARE REALIZATION FOR AN 8-BIT ‘k’

The Fast Point Multiplication (FPM) hardware realization is shown in Fig. 8. Here, Multiplexer \( MX_0 \) realizes \( Q_0 \) as given by Equation (45). \( MX_1 \) realizes \( Q_1 \) as given by Equation (44), \( MX_2 \) realizes \( Q_2 \) as given by Equation (43) and \( MX_3 \) realizes \( Q_3 \) as given by Equation (42). Point Adder \( PA_1 \) gives \( (Q_0 + Q_1) \) while \( PA_2 \) gives \( (Q_2 + Q_3) \). Finally, \( PA_3 \) gives \( (Q_0 + Q_1) + (Q_2 + Q_3) \) which is the output \( kp \) as given by Equation (47). Thus the hardware presented in Fig. 8. realizes \( kp \) using the precomputed products of \( P \) and 4x1 multiplexers.

#### 6.1. Timing Analysis of the FPM

In the FPM circuit of Fig. 8, all the 8 bits of the multiplier \( k \) are applied simultaneously to the multiplexers at say \( T_0 = 0 \). It is presumed that all the input signals to the multiplexers are readily available before \( T_0 \). Hence we have to calculate the time delay due to multiplexers and Point Adders. Compared to the running time of a Point Adder, the time delay in a multiplexer, which is a combinational circuit, is negligibly small. Therefore we neglect the delay in multiplexers and we assume that the outputs of the multiplexers \( Q_0, Q_1, Q_2 \) and \( Q_3 \) are available to the input of adders at \( T_0 = 0 \).

Let the input output transition time delay in the Point Adder \( PA_1 \) be \( A \) in appropriate time units. The Point Adders are similar in design and construction, and therefore time delays are also same. That is the input output transition time delay of each Point Adder is take as \( A \). For \( PA_1 \), the output \( (Q_0 + Q_1) \) would be available after a delay of \( A \). This can be expressed as Equation (49):

\[ T(Q_0 + Q_1) = A \]  

Similarly Equation (50):

\[ T(Q_2 + Q_3) = A \]
Fig. 7. $Q_0$ as the output of a 4×1 Multiplexer

Fig. 8. Fast Point Multiplication (FPM) module using multiplexers
Therefore the inputs \((Q_0 + Q_1)\) and \((Q_2 + Q_3)\) to \(PA_3\) are simultaneously available with a delay of \(A\). To this, we add the delay in \(PA_3\) to get the final delay as Equation (51):

\[
T\left((Q_0 + Q_1) + (Q_2 + Q_3)\right) = T(kP) = A + A = 2A \quad (51)
\]

Thus, for an 8 bit FPM unit, the delay is 2A.

### 6.2. Comparison with a Conventional Multiplier

Consider the Right-to-left binary method of Point Multiplication (De Dormole and Quisquater, 2007). With precomputation, the doubling time is eliminated and the running time is \(nA\) where \(n\) is the Hamming weight of \(k\), that is the number of 1’s in \(k\). When the size of \(k\) is 8 bits, the maximum value of \(n\) is 8. Therefore the worst case delay in the conventional method is 8A and the average case is 4A. In our method, the running time is 2A. Thus the speed of our method is twice that of the conventional method.

### 6.3. Hardware/Complexity

The inputs to each multiplexer are 4 elliptic curve points. Each point has two co-ordinates \((x, y)\). The maximum size of each component is given by \(m\), where

\[
m = \left\lfloor \log_2 P \right\rfloor \quad (52)
\]

Here \(p\) is the prime number of the prime field \(\mathbb{F}_p\). Therefore the size of each point is 2m. Hence the total number of signals at the input of each multiplexer will be \(4 \times (2m) = 8m\). The 0 input to the multiplexer Fig. 7 can be eliminated because it is a constant and zero. Hence, externally 3 inputs of size \(2m\) each have to be considered. Then the number of input signals would be \(3 \times (2m) = 6m\). For a 160 bit \(p\), the size of inputs to a multiplexer would be \(6 \times 160 = 960\) and the size of the output would be \(2 \times 160 = 320\).

### 6.4. Fast Point Multiplication Module

The Fast Point Multiplication shown in Fig. 8 along with the precomputation hardware can be represented by a modular block as shown in Fig. 9. The module is called as \(\text{FPM}_8\) which stands for Fast Point Multiplier Module for 8 bit sized \('k'\) that gives output \(kP\) with inputs \('P'\) and \('k'\). The module is shown in Fig. 9. The additional output \(2^3P\) is for concatenation.

### 7. CONCATENATION OF FAST POINT MULTIPLIER MODULES

When the size of \('k'\) is large, the 8-bit \(\text{FPM}_8\)'s can be concatenated to realize \(kP\) for large sized \(k\). In the realization shown in Fig. 10, the size of \('k'\) is 32 bits which is expressed in base 256 format as Equation (53):

\[
k = [K_3, K_2, K_1, K_0]_{256} \quad (53)
\]

Here, \(K_3, K_2, K_1\) and \(K_0\) are 8 bit each. \(K_0\) is the LSB and \(K_3\) is the MSB. The value of \(k\) is given by Equation (54):

\[
k = 256^3K_3 + 256^2K_2 + 256K_1 + K_0
\]

Therefore \(kP\) is given by Equation (55):

\[
kP = 256^3PK_3 + 256^2PK_2 + 256PK_1 + PK_0
\]

In our scheme, 256P, 256^2P, 256^3P and 256^4P are pre-computed and readily available as shown in Fig. 10. Let us designate these values by the symbols \(P_0, P_1, P_2\) and \(P_3\) as Equation (56-59):

\[
P_0 = P \quad (56)
\]
\[
P_1 = 256P \quad (57)
\]
\[
P_2 = 256^2P \quad (58)
\]
\[
P_3 = 256^3P \quad (59)
\]

Substituting these symbols in Equation (17) we get:

\[
kP = P_1K_3 + P_2K_2 + P_3K_1 + P_0K_0 \quad (60)
\]

Equation (60) is rewritten as:

\[
kP = (P_1K_3 + P_2K_2) + (P_3K_1 + P_0K_0) = S_2 + S_1
\]

Thus, \(kP\) is realized as the sum of \(S_1\) and \(S_2\) where:

\[
S_2 = (P_1K_3 + P_2K_2) \quad (62)
\]

and:

\[
S_1 = (P_3K_1 + P_0K_0) \quad (63)
\]

Equation (61-63) are realized using three point adders as shown in Fig. 10. In the circuit of Fig. 10, signals \(K_0, K_1, K_2\) and \(K_3\) are applied simultaneously.
Fig. 9. Fast Point Multiplier (FPM) Module

Fig. 10. Concatenation of FPM8’s for a 32 bit k
7.1. Latency of Signal kP for FPM₈ and FPM₃₂

From Equation (51), we know that the latency of each FPM₈ is 2A. Therefore the latencies of Pₓ,K₁ and Pₓ,K₀ are Equation (64):

\[ L(P,K_x) = L(P,K_0) = 2A \]  \hspace{1cm} (64)

To this, the latency of PA-1 is added to get Equation (65):

\[ L(S_x) = 2A + A = 3A \]  \hspace{1cm} (65)

Similarly, the latency of S₁ is Equation (66):

\[ L(S_1) = 3A \]  \hspace{1cm} (66)

Therefore the latency of kP = S₂ + S₁ is Equation (67):

\[ L(kP) = 3A + A = 4A \]  \hspace{1cm} (67)

Thus, for a 32 bit Fast Point Multiplier (FPM₃₂) the overall latency is 4A.

7.2. Extension of FPM’s to 128 Bits and 256 Bits

The hardware presented in Fig. 10 realizes kP for a 32 bit k. This circuit can be called FPM₃₂. Similar to the circuit of Fig. 10, four FPM₃₂’s can be concatenated to realize FPM₁₂₈ₚ which realizes kP with the size of ‘k’ equals 128 bits. The latency of this would be 4A + 2A = 6A. Similarly, four such FPM 128’s can be concatenated to get FPM₅₁₂ which can give out kP with a 512 bit k. Here, the latency would be 6A+2A = 8A.

7.3. Register Size Requirement for FPM₈ and FPM₃₂

In FPM₈, let the size of P be N-bits Equation (68-71):

Then the size of 256P will be N + 8  
\hspace{1cm} (68)

the size of 256²P will be N + 16  
\hspace{1cm} (69)

the size of 256³P will be N + 24  
\hspace{1cm} (70)

the size of 256⁴P will be N + 32  
\hspace{1cm} (71)

Here N is the NIST standard Value for ECC. Here also, the sizes will be increasing progressively. In the case of FPM₅₁₂ₚ, the register sizes are calculated similarly and implemented.

8. COMPARISON WITH OTHER METHODS

In our proposed hardware realization, a large number of PD’s and PA’s are used. Since the PD modules used are identical in design and characteristics, it is easy to replicate and integrate them. Similarly, PA modules can be replicated and integrated. This makes the Field Programmable Gate Array (FPGA) implementation of Elliptic curve point multiplication easy and efficient. This type of modular approach has not been attempted earlier. Same holds good for Controlled Buffers. In our method, the number of PD’s and PA’s used are m and (m-1) respectively which are relatively large. For example the NIST standard for m specifies one of the values from the set \{192, 224, 256, 384 and 521\}. In our method, all the bits of k are applied simultaneously. Thereby shifting of the bits of k one at a time is avoided (Kumar, 2006; Schinianakis et al., 2009; Portilla et al., 2010; Jacob et al., 2013). This saves ‘t’ clock cycles of time where ‘t’ is the size of ‘k’ in bits.

9. CONCLUSION

Two new theoretical hardware modules for Elliptic Curve Point Multiplication are described. PMMₘ’s and FPMₘ’s provide fast multiplication and they can be easily cascaded to realize point multiplication for larger values of k.

PMMₘ (w = 8) and FPMₘ use available Point Addition and Point Doubling sub modules, Therefore our proposed methods are faster compared to the conventional methods. Compared to the first method PMMₘ, the second method FPMₘ is faster, even though it requires more register space. From these modules we can create a macro model for realization of elliptic curve point multipliers for very large k. The techniques described can be modified for Point Multiplication over binary field.

These methods require large register spaces for storing the precomputed products of P as discussed in section 4.3 and 7.3. But the modules are similar and can be easily replicated.

Our proposed scalar multiplication modules are easily scalable and can be used independently or as sub modules in an elliptic curve crypto system.
In future, Fast Elliptic Curve Point Multiplication using Balanced Ternary Representation and Pre-computation over GF(p) can be investigated. The existing investigation can be extended to address varied design parameters like speed, power and area.

10. ACKNOWLEDGEMENT

The researcher would like to thank the Chairman Dr. R N Shetty, Director Dr. H N Shivashankar, Principal Dr. M K Venkatesha of RNS Institute of Technology for their constant support and encouragement and also to thank her professor N Bhaskara Rao, for his guidance and helpful comments in the study.

11. REFERENCES

De Dormole, G.M. and J.J. Quisquater, 2007. High-speed hardware implementations of elliptic curve cryptography: A survey. J. Syst. Archit., 53: 72-84. DOI: 10.1016/j.sysarc.2006.09.002

Ding, Q., T. Reece and W.H. Robinson, 2013. Timing analysis in software and hardware to implement NIST elliptic curves over prime fields. Proceedings of the 3rd International Workshop on Petri Nets and Performance Models, Aug. 4-7, IEEE Xplore Press, Columbus, pp: 1358-1362. DOI: 10.1109/MWSCAS.2013.6674908.

Ghosh S., M. Alam, I.S. Gupta and D.R. Chowdhury, 2007. A robust GF(p) parallel arithmetic unit for public key cryptography. Proceedings of the 10th Euromicro Conference on Digital System Design Architectures, Methods and Tools, Aug. 29-31, IEEE Xplore Press, Germany, pp: 109-115. DOI: 10.1109/DSD.2007.14

Hankerson, D., S. Vanstone and A.J. Menezes, 2004. Guide to Elliptic Curve Cryptography. 1st Edn., Springer, New York, ISBN-10: 038795273X, pp: 311.

Idrissi, Y.E.H.E., N. Zahid and M. Jedra, 2012. Security analysis of 3GPP (LTE)-WLAN interworking and a new local authentication method based on EAP-AKA. Proceedings of the International Conference on Future Generation Communication Technology, Dec. 12-14, IEEE Xplore Press, London, pp: 137-142. DOI: 10.1109/FGCT.2012.6476561

Ismail, E.S. and M.S. Hijazi, 2012. Development of a new elliptic curve cryptosystem with factoring problem. Am. J. Applied Sci., 9: 1443-1447.DOI: 10.3844/ajassp.2012.1443.1447

Jacob, N., S. Saetang, C.N. Chen, S. Kutzner and S. Ling et al., 2013. Feasibility and practicability of standardized cryptography on 4-bit micro controllers. Selected Areas Cryptography, 7707: 184-201. DOI: 10.1007/978-3-642-35999-6_13

Jie, L.K. and H. Kamarulhaili, 2011. Polynomial interpolation in the elliptic curve cryptosystem. J. Math. Stat., 7: 326-331. DOI: 10.3844/jmssp.2011.326.331

Kumar, S.S., 2006. Elliptic Curve Cryptography for Constrained Devices. Dissertation Submitted for the Award of PhD, Bochum, Germany.

Orlando, G. and C. Paar, 2001. A scalable GF(p) elliptic curve processor architecture for programmable hardware. Proceedings of the 3rd International Workshop Paris, May 14-16, Springer Berlin Heidelberg, France, pp: 348-363. DOI: 10.1007/3-540-44709-1_29

Portilla, J., A. Otero, E. de la Torre, T. Riesgo and O. Stecklina et al., 2010. Adaptable security in wireless sensor networks by using reconfigurable ECC hardware coprocessors. Int. J. Distribut. Sensor Networks. DOI: 10.1155/2010/740823

Schinianakis, D.M., A.P. Fournaris, H.E. Michail, A.P. Kakarountas and T.S. Stouraitis, 2009. An RNS implementation of an Fp elliptic curve point multiplier. IEEE Trans. Circ. Syst., 56: 1202-1213. DOI: 10.1109/TCSI.2008.208507

Shivkumar, S. and G. Umamaheswari, 2014. Certificate authority schemes using elliptic curve cryptography, RSA and their variants-simulation using ns2. Am. J. Applied Sci., 11: 171-179. DOI: 10.3844/ajassp.2014.171.179