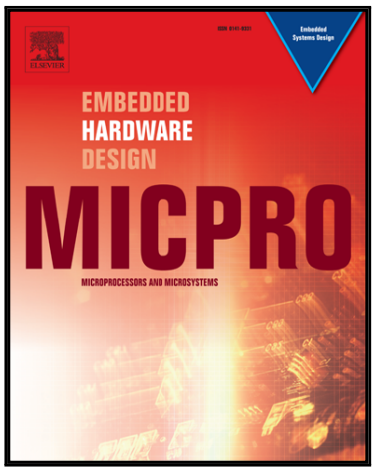

High-Speed FPGA Implementation of Full-Word Montgomery Multiplier for ECC Applications

Safiullah Khan, Khalid Javeed, Yasir Ali Shah

PII: S0141-9331(17)30284-3
DOI: https://doi.org/10.1016/j.micpro.2018.07.005
Reference: MICPRO 2721

To appear in: *Microprocessors and Microsystems*

Received date: 1 June 2017
Revised date: 6 June 2018
Accepted date: 17 July 2018

Please cite this article as: Safiullah Khan, Khalid Javeed, Yasir Ali Shah, High-Speed FPGA Implementation of Full-Word Montgomery Multiplier for ECC Applications, *Microprocessors and Microsystems* (2018), doi: https://doi.org/10.1016/j.micpro.2018.07.005

High-Speed FPGA Implementation of Full-Word Montgomery Multiplier for ECC Applications

Safiullah Khan[a], Khalid Javeed[b,*], Yasir Ali Shah[a]

[a] *Electrical Engineering Department, COMSATS Institute of Information Technology, Abbottabad, Pakistan*
[b] *Computer Engineering Department, Bahria University, Islamabad, Pakistan*

Abstract

Modular multiplication is the most crucial operation in many public-key crypto-systems, which can be accomplished by integer multiplication followed by a reduction scheme. The reduction scheme involves a division operation that is costly in terms of computation time and resource consumption both on hardware and software platforms. Montgomery modular multiplication is widely used to eliminate the costly division operation. This work presents an efficient implementation of full-word Montgomery modular multiplication. This incorporates the more efficient Karatsuba algorithm where the complexity of multiplication is reduced form $\mathcal{O}(n^2)$ to $\mathcal{O}(n^{1.58})$. The Karatsuba algorithm recommends to split the operands into smaller chunks. Two methods of operand splitting are exploited: 1) Four Parts (FP) Splitting and 2) Deep Four Parts (DFP) Splitting. These methods are then used in the design of Integer Multiplier (IM) and Montgomery Multiplier (MM). The design is synthesized using Xilinx ISE 14.1 Design Suite for Virtex-5, Virtex-6 and Virtex-7. Compared with the traditional implementations, the proposed scheme outperforms all other designs in terms of throughput and area-delay product. Moreover, the proposed scheme utilizes the least hardware resources in the known literature. The proposed MM design is able to compute modular multiplication for the Elliptic Curve Cryptography (ECC) field sizes of 192-512 bits.

Keywords: Montgomery modular multiplication, FPGA, Karatsuba algorithm

1. Introduction

Public Key Cryptography (PKC) [1], also called asymmetric cryptography is a cryptographic scheme which uses a pair of keys, namely the public key and the private key. As the name depicts the public key is known to everyone while the private key is only known to the receiver of the message. The keys are mathematically related but never the same. Unlike the symmetric key cryptography which uses the same key for encryption and decryption, here each key is assigned separate task. The public key encrypts the data while only the corresponding private key is used for decryption. Moreover the derivation of public key from the private key is computationally infeasible. This permits the exchange of the public key freely.

PKC algorithms most commonly include schemes which are based on RSA [2] and Elliptic Curve Cryptography (ECC) [3],[4]. ECC algorithms are based on the algebraic structures of the elliptic curves over the finite fields. All of the PKC schemes require arithmetic operations (modular addition/subtraction, modular multiplication and modular inversion/division) with large operands and the security of the schemes increase with the increased size of the operands [5]. For the same level of security ECC requires much smaller operands as compared to RSA [6]. This makes ECC a popular choice for implementing the PKC schemes, especially when used in resource constrained environments.

ECC algorithms can be implemented in software with high level of flexibility but at a cost of high time complexity. They can also be implemented in hardware with high running speed due to the efficient use of the modular multipliers. The ECC algorithm, as shown in the Figure 1, can be divided into four layers. The bottom layer consists of finite-field modular operations like addition, subtraction, multiplication and inversion. Among these modular operations, multiplication and inversion are most costly in terms of computation time and hardware resources. In particular modular inversion is very slow in hardware and software as compared to modular multiplication. However the modular inversion can be eliminated by introducing extra multiplications and by changing the coordinate system from affine to projective [7]. Hence an overall performance of the ECC algorithm is dependent on the modular multiplication if projective coordinates system is adopted. Therefore modular multiplier is the bottleneck in ECC based crypto-systems and its optimization is one of the common strategies to boost the performance of any ECC scheme.

Modular multipliers can be classified into three types in general. A standard method for computing a modular multiplication involves a division over a modulus M.

*corresponding author
Email address: khalid.buic@bahria.edu.pk (Khalid Javeed)

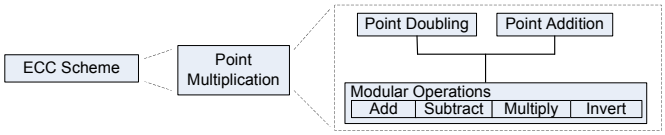

Figure 1: ECC Algorithm

The division over modulus M is an expensive operation in terms of computation time and resource consumption. Interleaved Modular Multiplication (IMM) as proposed by [8] and efficiently implemented by [9] and [10] is a method in which reduction is done while multiplication. The intermediate results are reduced and then added to get the final result. The most common method for a modular multiplication operation is the Montgomery Modular Multiplication (MMM) [11]. Different implementation possibilities of the Montgomery algorithm are discussed and analyzed in [12]. The IMM and MMM methods have the advantage of being flexible, which means that these can be used for any primes. However, the MMM method is significantly fast for the large operands as compared to the IMM method [13].

1.1. Related work

Many ideas have been proposed for the efficient hardware implementation of MMM. All these implementations can be broadly classified into two major categories, the bit-wise implementations and the block-wise implementations. The bit-wise implementations use the standard FPGA fabric and does not use any of the dedicated multipliers present in the modern FPGAs. The designs based on FPGA fabric tends to be slower when compared to the designs that utilize FPGA dedicated multipliers. The block-wise implementation divides the operands into chunks and these chunks are then multiplied by using the dedicated embedded multipliers. The designs based on embedded multipliers offer higher speed and can be adopted in time critical applications.

Several implementations that are using the dedicated multipliers and the fast carry chain adders are discussed. Mondal et al. in [14] proposed to use 64×64 bit soft cores for their design and different configurations and their hardware resources have been discussed. Brinci et al. in [15] designed a multiplier for Barreto-Naehrig (BN) curves. They have used non-standard splitting to fit the Xilinx Digital Signal Processor (DSP) block but this design can only be used for the BN curves defined over a special prime number and lacks generality. Kuang et al. in [16] proposed a low-cost high-performance Montgomery multiplication algorithm where Carry-Save Adder (CSA) has been exploited to avoid propagation delay at each addition operation. Configurable CSA was proposed to reduce extra clock cycles for pre-computation and format conversion. In [17] Rezai et al. designed an efficient Montgomery multiplication architecture based on digit serial computation. It relaxes high-radix partial multiplication to binary multiplication. Several multiplications of consecutive zero bits are performed in one clock cycle. Also right-to-left and left-to-right modular exponentiations architectures are modified to use as its structural unit. The design in [18] optimizes 512-bit addition and 256-bit multiplication using 64-bit carry chains and multiplier soft cores respectively and achieved a frequency of 188 MHz. Yang et al. in [19] achieved almost 50 % running efficiency compared with the traditional implementation method by efficiently using IP cores of Xilinx FPGA to design 512-bit addition and 256-bit multiplication. The design of a hardware elliptic curve cryptographic processor is presented in [20]. For the modular multiplication, full-word Montgomery modular multiplication along with the corresponding architectures are described. The 256-bit integer multiplier is designed by cascading 16-bit unsigned multipliers and this process is continued till the desired multiplier size is achieved. Addition is performed using the fast carry look-ahead adders. The design in [21] propose to split the operands in digits and then the computations are performed. The complexity of algorithm depends upon the size of digits and not on operand size. The hardware architecture of the proposed algorithm is also described.

Many designs have been proposed that employ Karatsuba algorithm to enhance the efficiency of Montgomery algorithm. Gong et al. in [22] designed a 256-bit Montgomery multiplier by utilizing the embedded 18×18 multipliers with a 5-stage pipeline structure. Operands are splitted into two parts according to the Karatsuba algorithm to reduce the number of embedded multipliers. This architecture has optimized the clock cycles however the operating frequency is limited to 30.38 MHz. The design in [23] utilizes nine 64-bit multiplier soft cores as basic building blocks to develop Karatsuba-based integer multiplier. The integer multiplier is then incorporated in the design of 256-bit Montgomery multiplier. Critical path delay is optimized at the cost of increased iterations. In [24] the operands are divided into chunks and then these chunks are multiplied either using the embedded multipliers or the fine-grained logic depending upon the length of the chunk of operands. However, large multipliers may face routing delays when operating at higher frequencies. The Karatsuba based Montgomery multiplier in [25] employs two parts splitting approach for operands splitting. Higher radix Montgomery multiplication algorithm is utilized since it makes use of embedded multiplier blocks, however the design is expensive in terms of area utilization. The Montgomery multiplier in [26] utilizes the Karatsuba algorithm with 4 levels of recursion. The operands are splitted into two chunks using two parts splitting approach. Each chunk is then again divided into two chunks in a recursive fashion till the length of chunks matches with the length of embedded DSP blocks.

All of the implementations discussed above split the operands into two chunks either recursively or non-recursively for the application of Karatsuba algorithm. Our proposed design differs from the designs discussed as

we first suggest to find the optimum point for the number of chunks and then apply the Karatsuba algorithm. The detailed view is provided in the next section. In this way the best results in terms of area and performance can be achieved.

The implementations that utilize LUTs instead of the dedicated multipliers are also discussed. A serial interleaved modular multiplier based on radix-4 Booth multiplication is presented in [27]. It is a LUT based design and does not consume any embedded multipliers. The design in [28] provides a hardware architecture for implementation of the parallel interleaved modular multiplier. This architecture consist of four processing elements that operate in parallel to perform the tasks assigned according to the algorithm. The design of Montgomery modular multiplier using the systolic array architecture is demonstrated in [29]. Systolic array consist of repeated structures called cells that operate in parallel to reduce the critical path delay. Authors in [10] incorporates the Montgomery power laddering technique along with the radix-4 serial multiplier to achieve 50 % reduction in the clock cycles.

2. Contribution

Modular multiplication operation is the basic operation in many public-key crypto-systems. The most widely used algorithm for the implementation of modular multiplication is the MMM algorithm. This work presents an efficient architecture for the implementation of Montgomery multiplication algorithm. To enhance the efficiency of Montgomery Multiplier (MM) Karatsuba-Ofman algorithm is employed. The Karatsuba algorithm recommends to split the operands into chunks in order to reduce the complexity of multiplication. Operands can be splitted into any number of chunks and then multiplied. As we split the operands further the size of chunks decreases and the number of multiplications increases. In this work we have find a balanced point to optimize the hardware resources and the computation time simultaneously. The operands are splitted into four parts and then this technique is applied recursively to get the deep four parts splitting of operands. So the two methods introduced here are termed as Four Parts (FP) splitting and Deep Four Parts (DFP) splitting. These methods are then used in the design of Integer Multiplier (IM) architectures and the MM architectures. Since the IM is the most important operation in the Montgomery multiplication algorithm so enhancing the speed of IM can increase the overall efficiency of MM. The design is synthesized using Xilinx ISE 14.1 Design Suite for Virtex-5, Virtex-6 and Virtex-7. The proposed MM design is able to compute modular multiplication for the common ECC field sizes of 192-512 bits. It provides the best results in terms of throughput and the area-delay product, so it can be efficiently used in elliptic curves and pairing based crypto-systems.

Algorithm 1: Montgomery Modular Multiplication

Input: X, Y, M, $n = \log_2 M$, $R = 2^n$
$M_1 = -M^{-1} \bmod R$
Output: $Z = X \times Y \times R^{-1} \bmod M$

1 $D \leftarrow X \times Y$
2 $E \leftarrow D \times M_1 \bmod R$
3 $Z \leftarrow (D + E \times M)/R$
4 if$Z > M$ then return $Z - M$
5 Else return$Z$

2.1. Montgomery Algorithm

Montgomery modular multiplication [11] is a method for computing fast modular multiplication, introduced by Peter L. Montgomery in 1985. Montgomery multiplication computes $X \times Y \bmod M$, where X and Y are positive integers and M is a large prime. Conventional approaches for computing the remainder use division operation which is costly. Montgomery multiplication replaces the costly division operation by simple shift and add operations, however this method works only on the numbers represented in Montgomery domain. Therefore, to take the advantage the operands are first transformed into the Residue Number System (RNS) domain before the operation and then the result is re-transformed after operation. The radix R is selected to be two to the power of word length and must be greater than Modulus. R and M must be relatively prime for the algorithm to run.

Given n bit positive number M (modulus) and two n bit operands X and Y where $0 < X, Y < M$, the result Z of the modular multiplication is $Z = X \times Y \bmod M$.

2.2. Karatsuba-Ofman Algorithm

The Karatsuba-Ofman algorithm reduces the complexity of multiplication by splitting the operands into smaller and equal chunks. The complexity of regular multiplication using the school-book method is $\mathcal{O}(n^2)$. The method discovered by Karatsuba and Ofman [30] i.e. by splitting operands, reduces the complexity to $\mathcal{O}(n^{1.58})$, where n is the length of operands to be multiplied. For the multiplication of two numbers, Karatsuba-Ofman algorithm recommends to split the operands into higher and lower chunks as follows.

$$X_1 = (x_{n-1}, ..., x_{\lceil n/2 \rceil})$$

$$X_0 = (x_{\lceil n/2 \rceil - 1}, ..., x_0)$$

$$Y_1 = (y_{n-1}, ..., y_{\lceil n/2 \rceil})$$

$$Y_0 = (y_{\lceil n/2 \rceil - 1}, ..., b_0)$$

The operands $X = \sum_{i=0}^{2n-1} x_i 2^i$ and $Y = \sum_{i=0}^{2n-1} y_i 2^i$ can be rewritten as

$$X = X_0 + 2^n X_1$$

$$Y = Y_0 + 2^n Y_1$$

Table 1: Comparison of Lengths and Number of Multipliers for Karatsuba and School-book Methods

	School-Book		Karatsuba	
	Length	M	length	M
2-Parts	N/2	4	N/2	3
3-Parts	N/3	9	N/3	6
4-Parts	N/4	16	N/4	10

The result for school-book method of multiplication is given below:

$$Z = X \cdot Y = X_0Y_0 + 2^n(X_0Y_1 + Y_0X_1) + 2^{2n}X_1Y_1 \quad (1)$$

Four multiplications which are half the size of original operands are required as shown in the equation above. This multiplication can be reformulated using the Karatsuba algorithm as given:

$$Z = X \cdot Y = X_0Y_0 + 2^n(X_1Y_1 + X_0Y_0 - (X_1 - X_0)(Y_1 - Y_0)) + 2^{2n}X_1Y_1 \quad (2)$$

Comparing equations 1 and 2 it can be seen that only three multiplications are needed in equation 2 and it saves one multiplication. The operands can be further splitted to perform the process recursively till a reasonable size of operands is reached. Reducing four multiplications to three also enhances the speed of multiplication.

2.3. Operands Splitting

Operands can be splitted into any number of chunks and then multiplied using the School-book or Karatsuba method. Table 1 shows the number of multipliers needed when operands are divided into two, three or four parts for both methods. As we split the operands further size of operands decreases and the number of multipliers increases and a few additions are introduced to generate the final result. Splitting operands further may not be suitable as the cost of overhead additions may overcome the area and time saved.

2.3.1. Two-parts Splitting

Operands can be splitted into two parts according to the Karatsuba algorithm. The effective part of the Karatsuba algorithm is the computation of two differences, $X_1 - X_0$ and $Y_1 - Y_0$, and then calculate the product $(X_1 - X_0)(Y_1 - Y_0)$. So the equation $(X_1Y_1 + X_0Y_0 - (X_1 - X_0)(Y_1 - Y_0))$ can save one multiplication, which is the essence of the Karatsuba algorithm.

Modern FPGA devices have DSP blocks which contain dedicated multipliers. Virtex-5 and Virtex-6 DSP blocks contains asymmetrical 18×25 signed multipliers. If n is the size of the embedded multiplier, this architecture can be used for $2n$ bit multiplication, using three embedded multipliers. This can also be generalized for any multiplication where the operands size is less than $2n$ using only three embedded multipliers. Earlier FPGAs had only embedded multipliers, but the modern DSP blocks contain internal adder as well making it possible to perform the required additions inside the DSP blocks.

Table 2: Estimated Number of DSP Blocks for Splitting Techniques and for the DFP splitting technique

Bit-length	4-Parts	3-Parts	2-Parts	DFPS
192-bit	110	72	90	106
224-bit	120	144	132	112
256-bit	120	222	132	112
384-bit	300	336	348	260
512-bit	560	696	693	496

2.3.2. Four-parts Splitting

The Karatsuba-Ofman algorithm can be applied recursively. Four-parts splitting can be obtained by the recursive application of two-parts splitting. The operands for the four-parts splitting are given:

$$X = X_0 + 2^nX_1 + 2^{2n}X_2 + 2^{3n}X_3$$

$$Y = Y_0 + 2^nY_1 + 2^{2n}Y_2 + 2^{3n}Y_3$$

The result of the school-book method for multiplication is given below

$$Z = X \cdot Y = X_0Y_0 + 2^n(X_1Y_0 + X_0Y_1) + 2^{2n}(X_2Y_0 + X_1Y_1 + X_0Y_2) + 2^{3n}(X_3Y_0 + X_2Y_1 + X_1Y_2 + X_0Y_3) + 2^{4n}(X_3Y_1 + X_2Y_2 + X_1Y_3) + 2^{5n}(X_2Y_3 + X_3Y_2) + 2^{6n}(X_3Y_3) \quad (3)$$

The equation above shows that 16 multiplications are required and the operands for the multiplications are one fourth of the original operands and therefore 16 DSP blocks are used if each multiplication is restricted in size to use only single DSP block. Now the same equation after the application of Karatsuba-Ofman algorithm reformulates to:

$$Z = X \cdot Y = P_{00} + 2^n(P_{11} + P_{00} - D_{10}) + 2^{2n}(P_{22} + P_{11} + P_{00} - D_{20}) + 2^{3n}(P_{33} + P_{00} + P_{22} + P_{11} - D_{30} - D_{21}) + 2^{4n}(P_{33} + P_{22} + P_{11} - D_{31}) + 2^{5n}(P_{33} + P_{22} - D_{32}) + 2^{6n}P_{33} \quad (4)$$

By comparing equation 3 and 4 it can be seen that the number of multiplications are reduced to 10 here with the operands size remained one fourth of the original operands. Number of DSP blocks are also reduced to 10 given that each multiplication is reduced to single DSP block. So here 6 multiplications are saved. The partial products and the difference equations are given:

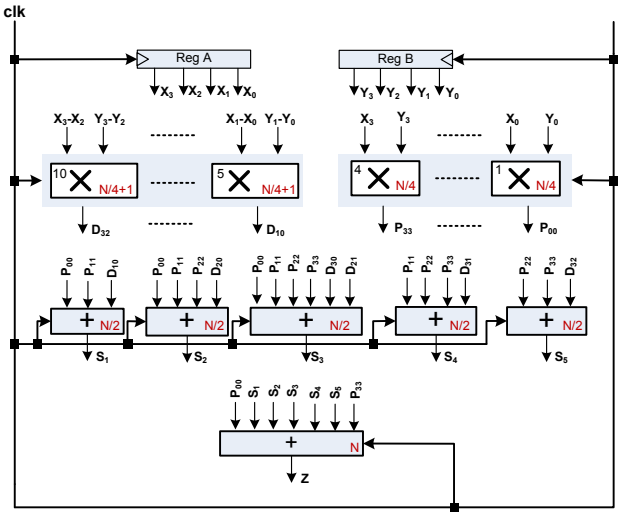

Figure 2: FP splitting Integer Multiplier

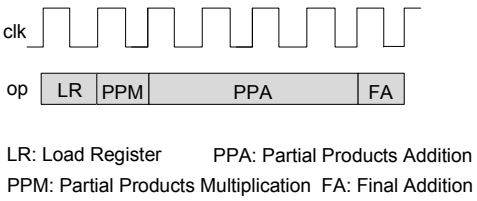

Figure 3: Clock cycle instants of the FP Splitting multiplier

$$P_{00} = X_0.Y_0$$
$$P_{11} = X_1.Y_1$$
$$P_{22} = X_2.Y_2$$
$$P_{33} = X_3.Y_3$$
$$D_{32} = (X_3 - X_2).(Y_3 - Y_2)$$
$$D_{31} = (X_3 - X_1).(Y_3 - Y_1)$$
$$D_{30} = (X_3 - X_0).(Y_3 - Y_0)$$
$$D_{21} = (X_2 - X_1).(Y_2 - Y_1)$$
$$D_{20} = (X_2 - X_0).(Y_2 - Y_0)$$
$$D_{10} = (X_1 - X_0).(Y_1 - Y_0)$$

Table 2 discusses the DSP blocks the device consumes for the design of the multipliers. Experimental results shows the number of DSP blocks for two, three and four parts which depicts that four parts splitting is the optimal choice. Splitting each chunk further into four parts, i.e. deep four parts splitting is also introduced in this work. Table 2 also shows the DSP blocks for DFP splitting technique. Comparing FP splitting with DFP splitting reveals that it can save hardware resources with large size operands.

Algorithm 2: Four Parts Splitting Multiplication Algorithm

Input: X, Y, $X = \sum_{i=0}^{3} 2^{ik} X_i$, $Y = \sum_{i=0}^{3} 2^{ik} Y_i$
Output: $Z = X \times Y$

for $i = 3; i \geq 0; i = i - 1$ **do**
  $j = i - 1$
  $P_{i,i} \leftarrow X_i \cdot Y_i$
  while $j \geq 0$ **do**
    $D_{i,j} \leftarrow (X_i - X_j) \cdot (Y_i - Y_j)$
    $j \leftarrow j - 1$
  end
end
$S_0 \leftarrow P_{00}$
$S_1 \leftarrow P_{11} + P_{00} - D_{10}$
$S_2 \leftarrow P_{22} + P_{11} + P_{00} - D_{20}$
$S_3 \leftarrow P_{33} + P_{00} + P_{22} + P_{11} - D_{30} - D_{21}$
$S_4 \leftarrow P_{33} + P_{22} + P_{11} - D_{31}$
$S_5 \leftarrow P_{33} + P_{22} - D_{32}$
$S_6 \leftarrow P_{33}$
$Z \leftarrow \sum_{i=0}^{6} 2^{ik} S_i$
return Z

2.4. FPGA Architecture

Field Programmable Gate Arrays (FPGAs) are the integrated circuits that can be configured by the user after manufacture. FPGA devices consist of Configurable Logic Blocks (CLB) which are connected through programmable interconnects. This property of the device has made it ideal for many applications including cryptography. Cryptographic systems such as secure communications, bank payments, data transfer have become an important part in our daily life. For the implementation of these systems FPGAs are the ideal choice. In addition to the configurable blocks, modern devices are provided with dedicated software cores and the hardware cores. Various soft cores for memory and arithmetic operations are available on the modern Xilinx FPGAs like Virtex-5, Virtex-6 and Virtex-7. These soft cores can easily be modified and multiple cores can be used. On the other hand CLBs gives the user the freedom for optimizing the code to increase the performance of the device.

3. Integer Multiplier

The efficiency of the MM is dependent on the efficiency of the IM. This paper employs the Karatsuba algorithm to enhance the efficiency of the multiplier. Two approaches have been exploited, the Four-Parts splitting and the Deep Four-Parts splitting. Each of them are discussed here along with the results.

3.1. Four-Parts Splitting Multiplier

Figure 2 shows the overall architecture for the FP splitting multiplier and Algorithm 2 explains the steps involved

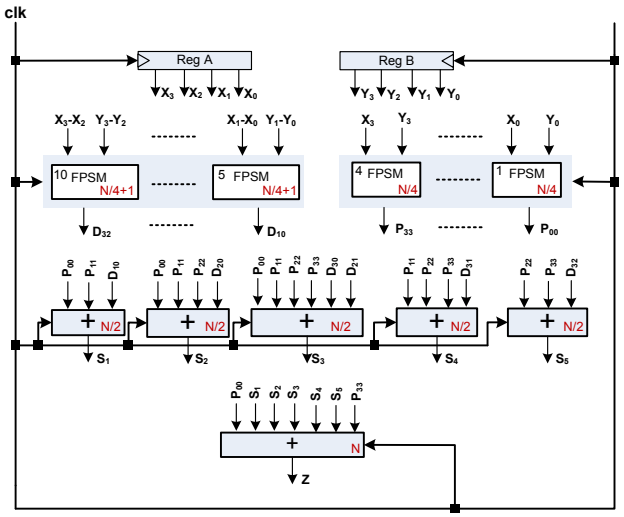

Figure 4: DFP Splitting Integer Multiplier

in the computation of the final result. In FP splitting method the operands to be multiplied are splitted into four equal chunks according to the Karatsuba-Ofman algorithm. At the start of multiplication the operands are stored in the input registers A and B as shown in the Figure 2. The next step is the generation of the partial products. Four unsigned multipliers of $N/4$ bits from 1 to 4 and six signed multipliers of $(N/4+1)$ bits from 5 to 10 generate the partial products. Ten multiplications in parallel are executed with the help of multipliers $^{1-10}\times$ as shown in the figure. Steps 1-7 of the algorithm explains the generation of the partial products. Ten multipliers are the result of the reduction achieved through Karatsuba-Ofman algorithm as shown in the equation 4. The results of the partial products acts as the inputs for the adders which are then added using $N/2$ bit fast carry chain adders as shown by the algorithm steps 9-14 and also shown in the figure which is followed by a final addition to generate the product. Here the splitting depth is 1.

The Figure 3 shows the clock cycles instants for the FP splitting multiplier. The whole multiplication takes seven clock cycles. During the first operation as shown in the figure, the input operands are loaded in the input registers. One clock cycle is needed for the operation Load Register (LR). The next operation comprises the computation of the partial products that is PPM (Partial Products Multiplication) during second operation as shown in the figure. The next operation adds the results of the PPM and consumes four clock cycles. PPA (Partial Products Addition) is the pipelined stage which adds the partial followed by the FA (Final Addition) of the products. FA consumes a single clock cycle. This as a whole consumes seven clock cycles for full operation.

3.2. Deep Four-Parts Splitting Multiplier

In DFP splitting method the splitting depth is 2 which means that each part of the operand is further splitted into four parts. One main advantage of DFP splitting multiplier is that the hardware is optimized. DFP splitting multiplier exploits the fact that the basic multiplier inside the DSP block in Virtex-6 can be fully utilized by splitting the operands till the point where the length of operands matches the size of multiplier in DSP block. The work in this paper uses 16×16 multiplier at the very deep level so that the design remain general for all the devices. Figure 4 explains the architecture of DFP splitting multiplier. The only difference is that the FP splitting multiplier is called instead of direct multiplication. Algorithm 3 explains the steps involved in the computation of the product. The operands are first splitted into four parts and each part is further splitted into four parts to achieve the deep four parts splitting. Steps 1-9 of the algorithm generate the partial products in parallel while the steps 12-16 performs the addition of the partial products, the results of which are then added by the fast carry chain adders.

Algorithm 3: Deep Four Parts Splitting Multiplication Algorithm

Input: X, Y, $X = \sum_{i=0}^{3} 2^{ik} X_i$, $Y = \sum_{i=0}^{3} 2^{ik} Y_i$
Input: $X_0 = \sum_{i=0}^{3} 2^{ik} X_{0,i} ... X_3 = \sum_{i=0}^{3} 2^{ik} X_{3,i}$
Input: $Y_0 = \sum_{i=0}^{3} 2^{ik} Y_{0,i} ... Y_3 = \sum_{i=0}^{3} 2^{ik} Y_{3,i}$
Output: $Z = X \times Y$
1 **for** $i = 3; i \geq 0; i = i - 1$ **do**
2 $\quad Z_i = FPS(X_i, Y_i)$
3 **end**
4 **for** $i = 3; i \geq 1; i = i - 1$ **do**
5 $\quad j = i - 1$
6 $\quad$ **while** $j \geq 0$ **do**
7 $\quad\quad D_{i,j} \leftarrow (X_i - X_j) \cdot (Y_i - Y_j)$
8 $\quad\quad j \leftarrow j - 1$
9 $\quad$ **end**
10 **end**
11 $S_0 \leftarrow Z_0$
12 $S_1 \leftarrow Z_1 + Z_0 - D_{10}$
13 $S_2 \leftarrow Z_2 + Z_1 + Z_0 - D_{20}$
14 $S_3 \leftarrow Z_3 + Z_0 + Z_2 + Z_1 - D_{30} - D_{21}$
15 $S_4 \leftarrow Z_3 + Z_2 + Z_1 - D_{31}$
16 $S_5 \leftarrow Z_3 + Z_2 - D_{32}$
17 $S_6 \leftarrow Z_3$
18 $Z \leftarrow \sum_{i=0}^{6} 2^{ik} S_i$
19 **return** Z

4. Overall Montgomery multiplier architecture

The Montgomery multiplication algorithm is shown in the Algorithm 1. This algorithm consist of three n bit integer multiplications which defines the overall efficiency of the algorithm. This work presents an efficient Montgomery multiplier architecture being implemented on modern FPGAs.

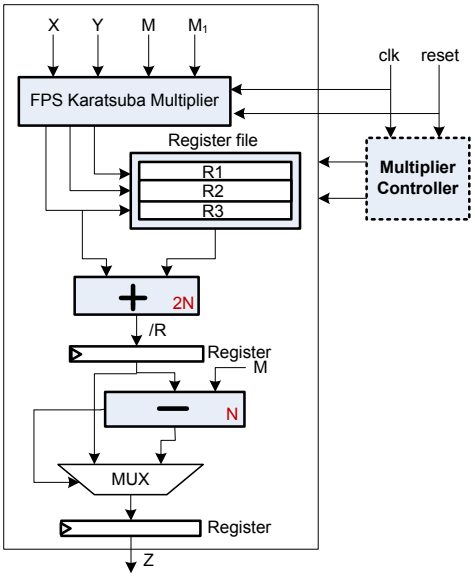

Figure 5: Montgomery modular multiplier

Figure 5 shows the proposed Montgomery multiplier architecture. It consist of n bit multiplier. The $2n$ bit registers are for holding the intermediate multiplication results, which are used in the later steps. The three multiplications are computed in series. The result of first multiplication is saved in the register which is then added to the result of third multiplication. A final reduction is used to compute the Montgomery result.

The proposed design performs the Montgomery multiplication by executing the series of operations as shown in the Figure 6. Registers X and Y are loaded with the operands on the first clock cycle as depicted by L, Load Register. During the first multiplication operation the operands are inputs X and Y. As soon as the multiplier gets the operands, first integer multiplication is started. It computes the product consuming 7 clock cycles and the $2n$ bit product is stored in the register during the operation W. It writes the result in the register file. The Figure 6 shows the clock cycle instants of the multiplier. Starting from the initial multiplication three multiplications are executed in series. The input registers X and Y are again loaded with the new operands during the operation Load Register. Operands for the second multiplication are modulus of the first multiplication result and M_1. 7 clock cycles are required for the multiplication. Again the results are stored in the registers which takes a single clock cycle. The input registers are loaded for the final multiplications. The operands here are modulus of the second multiplication result and M. 7 clock cycles are needed to complete the multiplication. In this way a total of 26 clock cycles are required to compute three multiplications in series. 3 more clock cycles are needed for the addition of the results of the multiplications according to the Algorithm 1 and the final comparison and subtraction if needed. In this way a total of 29 clock cycles are used to compute the whole Montgomery multiplication result.

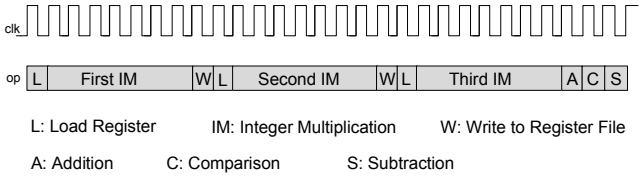

Figure 6: Clock cycle instants of the Montgomery multiplier

5. Implementation Results

The architectures proposed in the previous sections are not designed for any specific FPGA family. Length of the multiplier at the root level of DFP splitting multiplier is selected so that they can be implemented in several FPGA families.

Both the architectures FP splitting and DFP splitting multipliers are evaluated for five common operand sizes $p \in \{192, 224, 256, 384, 512\}$. The proposed Montgomery multiplier has been designed in Verilog HDL and implementation has been done in Xilinx ISE Design Suite 14.1 for Virtex-5, Virtex-6 and Virtex-7. Table 4 shows the implementation results for FP splitting and DFP splitting modular multipliers. As it is obvious that the deep four part splitting method gives the advantage of saving DSP blocks for some operand lengths, however it is advantageous only for higher operand lengths. The reason is that the chunk length when the splitting depth is 2, is less than the length of multiplier provided by the DSP block for the operands of smaller length. Implementation results also show that the total time of Montgomery multiplication is increased in case of DFP splitting technique. This is because it requires more clock cycles. Table 5 shows the performance comparisons with other designs implemented on similar platform.

The proposed design runs at 81.98 MHz and takes 29 clock cycles to compute the final Montgomery multiplication result for 256-bit operands. It consumes 120 DSP blocks. The design in [14] uses school-book method to compute the Montgomery modular multiplication. It consumes 16 64×64 soft cores and operates at 102 MHz. The hardware resources consumed are almost double than the proposed design while 24 % frequency overhead is achieved here. However, no information is provided about the time. Brinci et al. in design [15] exploits the asymmetric splitting of the operands to propose a design for the BN curves. Although the design is able to achieve a frequency of 208 MHz yet it is not flexible and is used for BN curves only, which is the major drawback. It also consumes 50 % more DSP blocks as compared to the proposed design. In [18] author presented the design for modular multiplier that is based on school-book method. The operating frequency is 188 MHz but the latency information and the overall time information is not provided. It also consumes almost 2 times of the DSPs than the proposed design. The design

Table 3: FP Splitting and DFP Splitting Multiplier

Platform	Bit-Length	LUTs	Slice Registers	Freq.(MHz)	CCs	DSPs	Min Period(ns)	Time(ns)
			FP Splitting Multiplier					
	192	4069	2794	130.94	7	110	7.63	53.41
	224	5210	2724	127.59	7	120	7.83	54.81
	256	5285	3108	124.42	7	120	8.03	56.21
	384	11659	4644	103.72	7	300	9.64	67.48
Virtex-7	512	20695	6180	99.68	7	560	10.03	70.21
			DFP Splitting Multiplier					
	192	4997	2613	86.51	10	130	11.55	111.50
	224	5925	3041	84.86	10	136	11.78	117.80
	256	6853	3468	83.27	10	136	12.00	120.00
	384	15211	5823	95.43	10	260	10.47	104.70
	512	23703	8122	86.39	10	492	11.57	115.70
			FP Splitting Multiplier					
	192	4069	2974	115.46	7	110	8.66	60.62
	224	5210	2724	112.35	7	120	8.90	62.30
	256	5285	3108	109.40	7	120	9.14	63.98
	384	11659	4644	92.30	7	300	10.83	75.81
Virtex-6	512	20695	6180	87.21	7	560	11.46	80.22
			DFP Splitting Multiplier					
	192	4997	2614	79.66	10	130	12.55	125.50
	224	5925	3041	77.98	10	136	12.82	128.20
	256	6853	3469	76.37	10	136	13.09	130.90
	384	15211	5817	94.12	10	260	10.62	106.20
	512	23703	8122	83.75	10	492	11.94	119.40
			FP Splitting Multiplier					
	192	4165	3071	91.10	7	110	10.97	76.79
	224	4650	2837	88.17	7	120	11.34	79.38
	256	5413	3237	85.43	7	120	11.70	81.90
	384	11851	4837	70.22	7	300	14.24	99.68
Virtex-5	512	20951	6437	65.65	7	560	15.23	106.61
			DFP Splitting Multiplier					
	192	5585	2710	61.33	10	110	16.30	163.00
	224	6609	3153	59.50	10	116	16.80	168.00
	256	7633	3597	57.78	10	116	17.30	173.00
	384	15399	6008	61.27	10	260	16.32	163.20
	512	23703	8122	54.80	10	492	18.24	182.40

in [19] proposed a 256-bit multiplier for Virtex-6 using the school-book method of multiplication. Both the factors, low operating frequency and high latency have been improved in our design which gives approximately 4 times better results in time. The operating frequency of this design is 40 MHz which is too low for high speed applications. Also our design has much better results in terms of the area-delay product. Authors in [20] have designed a multiplier for 256-bit operands and takes 700 ns for one modular multiplication which is almost 2 times more than proposed design. Although the implementations have been performed for a different platform so direct comparison is not possible, yet our design is better.

Chow et el. in [24] implemented a Montgomery modular multiplier that employs the Karatsuba algorithm along with deep pipeline stages. The number of these pipeline stages is not less than 18 while the exact number is not given in the paper. Also the time for modular multiplication is not provided. This architecture selects 32-bits as the limb width at the bottom layer of Karatsuba multiplier. Therefore, it requires 4 clock cycles to complete 128-bit addition and 8 clock cycles for 256-bit addition. So we estimate that the latency of this architecture is more than 50 cycles. Our proposed design takes 29 clock cycles to complete one modular multiplication. The design in [24] operates at higher frequency than our proposed design and performs a single modular multiplication in less time. However latency is an important parameter that can effect the actual performance in different applications like the scalar multiplication which is a basic operation in ECC applications. ECC scalar multiplication includes a series of modular multiplications. One scalar

Table 4: Proposed FP Splitting and DFP Splitting Montgomery Multiplier

Platform	Bit-Length	LUTs	Slice Registers	Freq.(MHz)	CCs	DSPs	Min Period(ns)	Time(ns)
FP Splitting Montgomery Multiplier								
	192	5157	4901	104.43	29	110	9.57	277.53
	224	6566	4745	102.80	29	120	9.73	282.17
	256	6688	5163	86.79	29	120	11.52	334.08
	384	13784	8489	74.00	29	300	13.51	391.79
Virtex-7	512	23511	11307	61.30	29	560	16.31	472.99
DFP Splitting Montgomery Multiplier								
	192	6085	4540	86.51	38	130	11.56	439.28
	224	7186	5288	84.86	38	136	11.78	447.64
	256	8287	6035	83.27	38	136	12.00	456.00
	384	17336	9659	85.49	38	260	11.69	444.22
	512	26519	13249	72.02	38	492	13.88	527.44
FP Splitting Montgomery Multiplier								
	192	4869	4267	94.47	29	110	10.58	306.82
	224	7462	4521	92.74	29	120	10.78	312.62
	256	6688	5163	81.98	29	120	12.19	353.51
	384	13784	8489	67.47	29	300	14.82	429.78
Virtex-6	512	23511	11307	55.00	29	560	18.18	527.22
DFP Splitting Montgomery Multiplier								
	192	6085	4541	79.66	38	130	12.55	476.90
	224	7186	5288	77.98	38	136	12.84	487.92
	256	8287	6036	76.37	38	136	13.09	497.42
	384	17336	9659	77.62	38	260	12.88	489.44
	512	26519	13249	60.88	38	492	16.42	623.96
FP Splitting Montgomery Multiplier								
	192	5602	4998	72.66	29	110	13.76	399.04
	224	5557	4858	63.05	29	120	15.85	459.65
	256	7324	5804	58.37	29	120	17.13	496.77
	384	14708	8489	44.98	29	300	22.22	644.38
Virtex-5	512	23511	11307	36.59	29	560	27.32	792.28
DFP Splitting Montgomery Multiplier								
	192	7022	4637	61.33	38	110	16.30	619.40
	224	8282	5400	59.50	38	116	16.80	638.40
	256	9544	6164	57.78	38	116	17.30	657.40
	384	18256	9852	52.61	38	260	19.00	722.00
	512	27768	13574	40.27	38	492	24.82	943.16

multiplication requires 1280 modular multiplications with data-dependence. Hence our proposed design will consume 37120 clock cycles when computing the scalar multiplication as compared to this design that takes 64000 clock cycles so our proposed architecture can perform better in ECC applications. Furthermore our design is better in terms of area-delay product. The design in [26] implemented a modular multiplier based on the Karatsuba algorithm. This design operates at a higher frequency than our proposed design and consumes the same number of clock cycles, hence has better results in time. However, our proposed design is better in terms of hardware resources utilized and area-delay product. The area-delay product shows that the proposed design is better optimized when both area and time are considered.

The results of the bit-wise implementations of the Montgomery modular multiplications are also discussed. As already stated that the bit-wise implementations use the standard FPGA fabric and does not make use of any dedicated multipliers which can be seen from the implementation results. The design in [10] used interleaved modular multipliers for 224-bit and 256-bit operands running at frequencies of 99 MHz and 96 MHz respectively. Time for Montgomery multiplication and throughput have been provided which shows that the proposed design is approximately 4 times better. A serial interleaved multiplier have been proposed in [27] performing multiplication for 192-bit, 224-bit and 256-bit. Implementation results are shown in the table. For 256-bit, the proposed design proves to be 4 times better. Similar work has been proposed in [28] using parallel interleaved multiplier. The overall time period have been approximately 2.5 times better. Also our

Table 5: Performance Comparisons

Design	Device	Bit-Length	DSP	Freq. (MHz)	Latency	Time (ns)	Mbps	LUT (k)	$A_L \times T$
Proposed FP Splitting Montgomery Multiplier									
[Proposed]	Virtex-6	192	110	94.47	29	306	627	4.9	1.50
		224	120	92.74		312	717	7.5	2.34
		256	120	81.98		353	725	6.7	2.36
Block-wise Implementations									
[14]	Virtex-6	256	256	102.67	-	-	-	-	-
[15]	Virtex-6	258	176	208	-	-	-	-	-
[18]	Virtex-6	256	256	188	-	-	-	-	-
[19]	Virtex-6	256	256	40.06	50	1248	205	24	29.9
[20]	Virtex-2 Pro	256	256	45.68	32	700	316	1.42	-
Block-wise implementations employing Karatsuba algorithm									
[24]	Virtex-6	256	108	336	50	160	-	17.0	2.72
[26]	Virtex-6	256	108	205.76	29	142	-	22.5	3.19
Bit-wise Implementations									
[10]	Virtex-6	224	-	99	-	1130	198	3.4	3.84
		256	-	96	-	1300	196	3.9	5.07
[27]	Virtex-6	192	-	-	98	967	198	3.0	2.90
		224	-	-	114	1160	193	3.4	3.94
		256	-	-	130	1360	188	3.9	5.30
[28]	Virtex-6	256	-	166	-	790	324	6.3	4.97

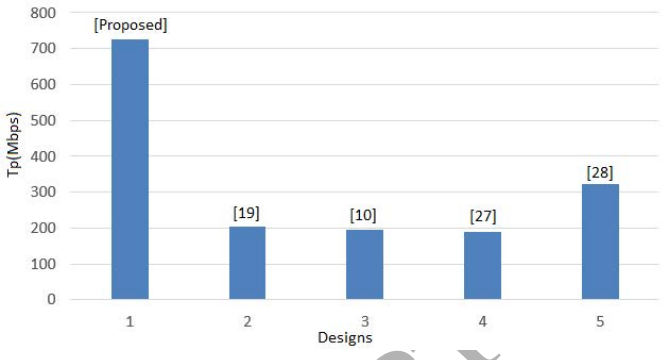

Figure 7: Throughput Comparisons for 256-bit Montgomery multipliers for Virtex-6

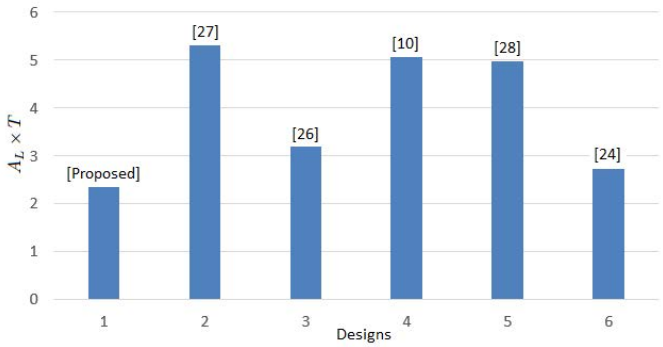

Figure 8: Area-delay Product Comparison for 256-bit Montgomery multipliers for Virtex-6

proposed design is better in terms of throughput and the area-delay product when compared with all the designs discussed above.

The throughput in terms of mega bits per second (Mbps) is an important performance metric. The graph in the Figure 7 shows the throughput comparisons for 256-bit Montgomery multipliers for Virtex-6 FPGAs. The throughput for the proposed design is 725 Mbps which is more than any of the designs presented here for the targeted device.

Similarly the Figure 8 shows the area-delay product of the proposed design in comparison with other designs for 256-bit Montgomery multipliers. The area-delay product for our proposed design is 2.36 which is less than any of the designs presented here.

The power consumption of the proposed design is also estimated using XPower Analyzer tool available in Xilinx ISE Design Suite 14.1. It is found that the power consumption of the proposed multiplier for Virtex-6 is estimated to be 6.402 W and 4.447 W for the bit-length of 256 and 224 respectively. These values are calculated for the default operating parameters of the device such as temperature at 25°C, Vccint at 1 V and Vccaux and Vcco at 2.5 V.

6. Conclusion

Montgomery modular multiplication plays an important role when implementing cryptographic algorithms on hardware platform. This paper presents an efficient implementation of full-word Montgomery modular multiplication which is suitable to apply to ECC and RSA cryptographic algorithms to promote their running speed.

This work exploits the efficiency of Karatsuba-Ofman algorithm to achieve a 256-bit multiplication using 120 DSP blocks. Two techniques FP splitting and DFP splitting have been designed to apply for the multiplication of the sizes $s = \{192, 224, 256, 384, 512\}$. Implementation results and comparisons show that the proposed scheme is better than other schemes. The proposed Montgomery multiplier is highly flexible as well and can be efficiently employed in cryptographic processor for elliptic curves and pairing based cryptography.

References

References

[1] W. Diffie, M. Hellman, New directions in cryptography, IEEE transactions on Information Theory 22 (6) (1976) 644–654.

[2] R. L. Rivest, A. Shamir, L. Adleman, A method for obtaining digital signatures and public-key cryptosystems, Communications of the ACM 21 (2) (1978) 120–126.

[3] V. S. Miller, Use of elliptic curves in cryptography, in: Conference on the theory and application of cryptographic techniques, Springer, 1985, pp. 417–426.

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

[5] D. J. Bernstein, T. Lange, Faster addition and doubling on elliptic curves, in: International Conference on the Theory and Application of Cryptology and Information Security, Springer, 2007, pp. 29–50.

[6] A. K. Lenstra, E. R. Verheul, Selecting cryptographic key sizes, Journal of cryptology 14 (4) (2001) 255–293.

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

[8] G. R. Blakely, A computer algorithm for calculating the product ab modulo m, IEEE Transactions on Computers 5 (C-32) (1983) 497–500.

[9] K. Javeed, X. Wang, Radix-4 and radix-8 booth encoded interleaved modular multipliers over general f p, in: Field Programmable Logic and Applications (FPL), 2014 24th International Conference on, IEEE, 2014, pp. 1–6.

[10] K. Javeed, X. Wang, M. Scott, Serial and parallel interleaved modular multipliers on fpga platform, in: Field Programmable Logic and Applications (FPL), 2015 25th International Conference on, IEEE, 2015, pp. 1–4.

[11] P. L. Montgomery, Modular multiplication without trial division, Mathematics of computation 44 (170) (1985) 519–521.

[12] C. K. Koc, T. Acar, B. S. Kaliski, Analyzing and comparing montgomery multiplication algorithms, IEEE micro 16 (3) (1996) 26–33.

[13] K. Javeed, D. Irwin, X. Wang, Design and performance comparison of modular multipliers implemented on fpga platform, in: International Conference on Cloud Computing and Security, Springer, 2016, pp. 251–260.

[14] A. Mondal, S. Ghosh, A. Das, D. R. Chowdhury, Efficient fpga implementation of montgomery multiplier using dsp blocks, in: Progress in VLSI Design and Test, Springer, 2012, pp. 370–372.

[15] R. Brinci, W. Khmiri, M. Mbarek, A. B. Rabaâ, A. Bouallegue, F. Chekir, Efficient multiplier for pairings over barreto-naehrig curves on virtex-6 fpga., IACR Cryptology EPrint Archive 2013 (2013) 5.

[16] S.-R. Kuang, K.-Y. Wu, R.-Y. Lu, Low-cost high-performance vlsi architecture for montgomery modular multiplication, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 24 (2) (2016) 434–443.

[17] A. Rezai, P. Keshavarzi, High-throughput modular multiplication and exponentiation algorithms using multibit-scan–multibit-shift technique, IEEE Transactions on Very Large Scale Integration (VLSI) Systems 23 (9) (2015) 1710–1719.

[18] K. Javeed, X. Wang, Efficient montgomery multiplier for pairing and elliptic curve based cryptography, in: Communication Systems, Networks & Digital Signal Processing (CSNDSP), 2014 9th International Symposium on, IEEE, 2014, pp. 255–260.

[19] Y. Yang, C. Wu, Z. Li, J. Yang, Efficient fpga implementation of modular multiplication based on montgomery algorithm, Microprocessors and Microsystems 47 (2016) 209–215.

[20] C. J. McIvor, M. McLoone, J. V. McCanny, Hardware elliptic curve cryptographic processor over $rmgf(p)$, IEEE Transactions on Circuits and Systems I: Regular Papers 53 (9) (2006) 1946–1957.

[21] M. Morales-Sandoval, A. Diaz-Perez, Scalable gf (p) montgomery multiplier based on a digit–digit computation approach, IET Computers & Digital Techniques 10 (3) (2016) 102–109.

[22] Y. Gong, S. Li, High-throughput fpga implementation of 256-bit montgomery modular multiplier, in: Education Technology and Computer Science (ETCS), 2010 Second International Workshop on, Vol. 3, IEEE, 2010, pp. 173–176.

[23] S. Ghosh, I. Verbauwhede, D. Roychowdhury, Core based architecture to speed up optimal ate pairing on fpga platform, in: International Conference on Pairing-Based Cryptography, Springer, 2012, pp. 141–159.

[24] G. C. Chow, K. Eguro, W. Luk, P. Leong, A karatsuba-based montgomery multiplier, in: Field Programmable Logic and Applications (FPL), 2010 International Conference on, IEEE, 2010, pp. 434–437.

[25] I. San, N. At, Improving the computational efficiency of modular operations for embedded systems, Journal of Systems Architecture 60 (5) (2014) 440–451.

[26] X. Yan, G. Wu, D. Wu, F. Zheng, X. Xie, An implementation of montgomery modular multiplication on fpgas, in: Information Science and Cloud Computing (ISCC), 2013 International Conference on, IEEE, 2013, pp. 32–38.

[27] K. Javeed, X. Wang, M. Scott, High performance hardware support for elliptic curve cryptography over general prime field, Microprocessors and Microsystems 51 (2017) 331–342.

[28] K. Javeed, X. Wang, Low latency flexible fpga implementation of point multiplication on elliptic curves over gf (p), International Journal of Circuit Theory and Applications 45 (2) (2017) 214–228.

[29] S. B. Ors, L. Batina, B. Preneel, J. Vandewalle, Hardware implementation of an elliptic curve processor over gf (p), in: Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on, IEEE, 2003, pp. 433–443.

[30] A. Karatsuba, Y. Ofman, Multiplication of many-digital numbers by automatic computers, in: Doklady Akad. Nauk SSSR, Vol. 145, 1962, p. 85.

Safiullah Khan

Khalid Javeed

Yasir Ali Shah

Safiullah Khan has completed his MS in Electrical Engineering from COMSATS Institute of Information Technology, Abbottabad, Pakistan in 2017. He has done Bachelors in Electronic Engineering from University of Engineering and Technology, Peshawar, Pakistan. His research interests include finite field arithmetic, information security and cryptographic engineering.

Yasir Ali Shah is pursuing his PhD from Electrical Engineering Department, COMSATS Institute of Information Technology, Abbottabad, Pakistan. He has done MSc in Electrical Engineering from Linkoping University, Sweden, in 2010. His research interests include Digital system design, VLSI architectures and Hardware design of cryptographic algorithms.

Khalid Javeed completed his PhD from School of Electronic Engineering, Dublin City University, Dublin, Ireland in 2016. He has done MSc in Electrical Engineering from Linkoping University, Sweden, in 2010. His research interests include computer architecture, finite field arithmetic and VLSI architectures for cryptographic primitives.