Hardware Implementation of Efficient Elliptic Curve Scalar Multiplication using Vedic Multiplier

Rakesh K. Kadu[1] and Dattatraya S. Adane[2]

[1, 2]Shri Ramdeobaba College of Engineering and Management, Department of Information Technology, Nagpur, India
[1]Yeshwantrao Chavan College of Engineering, Department of Computer Technology, Nagpur, India

Abstract: This paper presents an area efficient and high-speed FPGA implementation of scalar multiplication using a Vedic multiplier. Scalar multiplication is the most important operation in Elliptic Curve Cryptography(ECC), which used for public key generation and the performance of ECC greatly depends on it. The scalar multiplication is multiplying integer k with scalar P to compute Q=kP, where k is private key and P is a base point on the Elliptic curve. The scalar multiplication uses underlying finite field arithmetic operation i.e. addition multiplication, squaring and inversion to compute Q. From these finite field operations, multiplication is the most time-consuming operation, occupy more device space and it dominates the speed of Scalar multiplication. This paper presents an efficient implementation of finite field multiplication using a Vedic multiplier. The scalar multiplier is designed over Galois Binary field $GF(2^{233})$ for field size=233-bit which is secured curve according to NIST. The performances of the proposed design are evaluated by comparing it with Karatsuba based scalar multiplier for area and delay. The results show that the proposed scalar multiplication using Vedic multiplier has consumed 22% less area on FPGA and also has 12% less delay, than Karatsuba, based scalar multiplier. The scalar multiplier is coded in Verilog HDL, synthesizes and simulated in Xilinx 13.2 ISE on Virtex6 FPGA.

Keywords: Elliptic curve cryptography, Scalar multiplication, Karatsuba multiplier, Vedic multiplier, FPGA.

1. Introduction

Elliptic Curve Cryptography is public key cryptography proposed by Miller and Koblitz in 1985. ECC is gaining acceptance for implementing security standards in place of well-known RSA, DES cryptography algorithm. In ECC smaller key size provides more security i.e. 160-bit key provides the same security level compare with the 1024-bit key of RSA. Due to the above feature, this cryptosystem is suitable for devices, which is having less computation power, limited storage, and limited battery backup. Following Elliptic curve cryptosystem protocols are available for Key generation, Key exchange, Digital Signature, and data encryption;

- Elliptic Curve Diffie Hellman (ECDH)
- Elliptic Curve Digital Signature Algorithm (ECDSA)
- Elliptic Curve Integrated Encryption System(ECIES)

In the above ECC protocols, scalar multiplication is used for a public key generation at the sender and receiver end. The performance of the ECC protocol greatly depends on the efficient implementation of the scalar multiplication operation.

In Elliptic Curve Cryptography scalar multiplication is the most focused part for the optimization. Many authors have proposed different techniques for optimizing scalar multiplication operation. According to literature, optimization can be achieved at a different level of scalar computation. The author has proposed different approaches for optimizing scalar multiplication In the first approach the author has presented several techniques to reduce the number of iteration of computing point addition and point doubling operation. This can be achieved by representing the scalar k, in such a way that it reduces the hamming weight of inter k. This will reduce the number of iteration of point addition and point doubling operation.

In [1][2] the author presents methods based on this approach which is discussed in section 2.

In the second approach, the optimization can be done at the bottom level by the fast and efficient implementation of underlying finite field operation such as addition, multiplication, squaring and inversion. From the above finite field operation, multiplication, inversion is the most time-consuming operation and it also occupies more device space.

In [3] the author have proposed and implemented finite field multiplier using Binary, Simple, General and Hybrid Karatsuba multiplier over the projective coordinate system. The result shows that the Hybrid Karatsuba multiplier is more area efficient than other design.

In [4] Elliptic Curve scalar multiplier architecture for field size 163-bit has presented, the delay is reduced by adopting the pipeline strategy to implement point addition, point doubling, and Karatsuba multiplier. The architecture uses 3, 4 stage pipelining for ECSMA.

In [5] author proposed high speed architectures to implement point multiplication on binary Edwards and generalized Hessian curves. They perform a data-flow analysis and investigate maximum number of parallel multipliers to be employed to reduce the latency of point multiplication on these curves. Then, they modify the addition and doubling formulations and employ a newly proposed digit-level hybrid-double Gaussian normal basis multiplier to remove the data dependencies and hence reduce the latency of point multiplication. The results show that the proposed schemes improve the performance of point multiplication on binary Edward and generalized Hessian curves.

In [6] author has proposed a Hybrid Karatsuba Multiplier which requires least amount of space on a FPGA. Comparison shows that the proposed Hybrid Karatsuba Multiplier requires lesser number of gates compared with the conventional approach.

In [7] author has reorganized and reordered the critical path of the Lopez-Dahab scalar point multiplication architecture such that logic structures are implemented in parallel and operations in the critical path are diverted to noncritical paths. The proposed design is more area and delay efficient than existing design.

In[8] author proposed new high-speed pipelined application-specific instruction set processor (ASIP). Different levels of pipelining were applied to the data path to explore the resulting performances and find an optimal pipeline depth. Three complex instructions were used to reduce the latency by reducing the overall number of instructions, and a new combined algorithm was developed to perform point doubling and point addition using the application specific instructions. This approach reduces the overall delay of scalar multiplication.

In [9] author presents an implementation of Elliptic Curve components i.e. Point addition and Point Doubling over binary field GF(2^{233}) and presented the device utilization of the design.

In [10] author A high-performance architecture of elliptic curve scalar multiplication based on the Montgomery ladder method over finite field GF(2^m) is proposed. A pseudo pipelined word-serial finite field multiplier, with word size w, suitable for the scalar multiplication is also developed.

A high-performance scalar multiplication scheme based on the Montgomery scalar multiplication algorithm has been proposed. The proposed scheme has been optimized to be dependent only to the number of FF multiplications. It hides the multiplier's overhead and prevents a pipeline multiplier from stalling.

The proposed scheme performs a scalar multiplication in 25(m-1) clock cycles, which is approximately 2.75 times faster than a straightforward implementation and 1.6 times faster than the best implementations reported in this category in the open literature.

In [11] author reduces the computational complexity by proposing an novel modified Lopez-Dahab scalar point multiplication and left-to-right algorithms for point multiplication operation. Further bit-serial Galois-field multiplication is used in order to decrease hardware complexity and field multiplication operations are performed in parallel to improve system latency. The result, shows it reduces hardware costs, while the total time required for point multiplication is kept to a reasonable amount.

In this paper, we have proposed the finite field multiplication operation using a Vedic multiplier for scalar multiplication. For performance evaluation of the proposed scheme, we have implemented scalar multiplication using Hybrid Karatsuba multiplier and comparative analysis of multiplier are presented for area and delay. The scalar multiplier is coded using Verilog HDL and implemented on Virtex6 FPGA in Xilinx 13.2 ISE.

In the rest of the paper, Section 2 presents the detailed working of Scalar multiplication. Section 3 presents the mathematical background of Karatsuba and Vedic multiplier. FPGA Implementation of scalar multiplication for binary field $GF(2^{233})$ is discussed in Section 4. In section 5 we have discussed the implementation results. Section 6 presents the conclusion.

2. Scalar Multiplication

Scalar multiplication is one of the most important operations and basic operation in Elliptic curve cryptography[12]. The scalar multiplication is computing $Q=kP$, where k is an integer and $P(x_1,y_1)$ and $Q(x_2,y_2)$ are the points on an Elliptic curve.

The scalar multiplication has the form;

$$Q=kP \tag{1}$$

This can be calculated by adding point P exactly k-1 times itself which is shown in equation 2.

$$Q= P+P+\ldots\ldots\ldots+ P\ (k\ times) \tag{2}$$

The security of ECC depends on the difficulty of Discrete Logarithm Problem (DLP), which is finding k from given P and Q. Practically it is very difficult to find k if P and Q are known.

Scalar multiplication plays an important role in computing public key Q. It is a multiplication of integer k with scalar P. Figure 1 shows the layer model of scalar multiplication for computing Q.

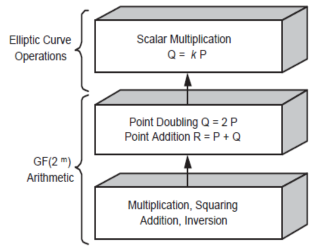

Elliptic Curve Operations
Scalar Multiplication Q = k P
Point Doubling Q = 2 P
Point Addition R = P + Q
GF(2 m) Arithmetic
Multiplication, Squaring Addition, Inversion

Figure 1. Layer model for Elliptic curve scalar multiplication.

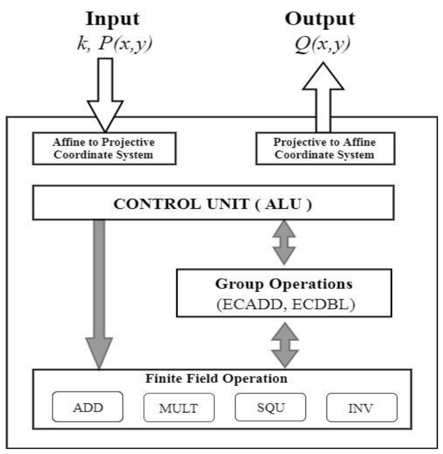

Input
k, P(x,y)
Output
Q(x,y)
Affine to Projective Coordinate System
Projective to Affine Coordinate System
CONTROL UNIT (ALU)
Group Operations (ECADD, ECDBL)
Finite Field Operation
ADD
MULT
SQU
INV

Figure 2. Architecture of ECC scalar multiplication

In this paper, we have proposed new Scalar multiplier architecture using Vedic mathematics which is shown in Figure 2. The ECC Scalar architecture consists of the Control unit (ALU) which calls ECADD and ECDBL. ECADD and ECDBL operation computes new Q(x,y) coordinate values using arithmetic instructions and finite field arithmetic operations.

The scalar multiplication calls point addition and points doubling repeatedly based on k for computing Q. While computation integer k is represented in binary representation; the multiplication cost depends on the length of k and the number of 1's in binary representation. In binary representation, if the bit is 1 then point addition and point doubling perform. If the bit is 0 then we only perform point doubling operation. Representing *k*, in such a way that it reduces 1'ˢ reduces the number of point doubling and addition operation, it improves the speed of scalar multiplication.

This can be achieved by reducing the hamming weight of the *k.* In [1] the author presents the following scalar multiplication methods and evaluated the computational cost for each method:

- Left to right Binary Method
- Right to left Binary method
- Non-Adjacent Form(NAF)
- Window-NAF method (wNAF)
- Mutual Opposite Form(MOF)
- Shamir Method–Parallel Computation
- Joint Sparse Form(JSF)
- Direct Doubling
- Double Base Number System (DBNS)
- Sliding NAF method
- Binary Windowing method
- Montgomery method

Algorithm1 shows the steps to compute *Q* using Point doubling and Point Addition operation[1].

```
Algorithm1: ECC Scalar multiplication using
Left to Right Binary
Input:
        Integer k ≠ 0 , $(k_{n-1}, …,k_1, k_0)_2$
        Base point $P(x_1,y_1)$ ϵ E.
Output: Q=kP
Begin
  Q = O
     for i= n-2 downto 0 do
        Q=2Q ( Point Doubling: ECDBL)
      if (k=1) then
        Q=P+Q  (Point Addition:ECADD)
    end
     end
  Return(Q)
end
```

For a given point $P(x_1,y_1)$, $Q(x_2, y_2)$ on an Elliptic curve, Point Addition (ECADD) and Point Doubling (EDDBL) of scalar multiplication are computed using following formulas, resulting in new point $R=(x_3,y_3)$ in the Affine coordinate system.

$$\text{ECADD: if } P\neq Q \qquad (3)$$
$$x_3=\lambda^2-x_1-x_2$$
$$y_3=\lambda(x_1-x_3)-y_1$$
$$\lambda= y_2-y_1/x_2-x_1$$
$$\text{ECDBL: if } P=Q \qquad (4)$$
$$x_3=\lambda^2-2x_1$$
$$y_3=\lambda(x_1-x_3)-y_1$$
$$\lambda = 3x_1^2+a/2y_1$$

If $P\neq Q$ then points addition will perform, and if $P=Q$, then point doubling operation will be called. The result of point addition or doubling results a new points *R* will always be another point on the Elliptic curve. Figure 3 shows point addition and Figure 4 shows the doubling operation on the elliptic curve to get third point R on the curve.

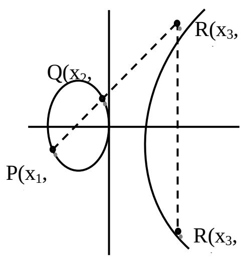


Figure 3. Point addition: R=P+Q.

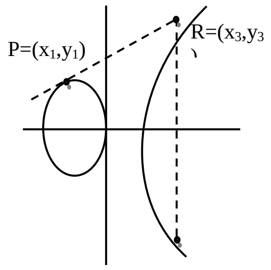


Figure 4. Point doubling: R=2P.

3. Karatsuba and Vedic Multiplier for Finite Field Multiplication

Multiplier plays a vital role in digital circuit design. Among all the arithmetic operation, multiplication is the most expensive operation. The computational time for multiplication depends on the size of multiplier and multiplicand. For large numbers, the naïve multiplier is not suitable. In digital design, different multipliers i.e Array, Booth, Wallace- Tree[13], Dadda, and Karatsuba are used for performing the multiplication operation. In [6][14] the author has analyzed different multipliers and its variations for their performance. In this section, we will present the working of Karatsuba multiplier and a Vedic multiplier.

3.1 Karatsuba multiplier

The Karatsuba multiplier works on divide and conquers method for multiplying two numbers. The Karatsuba multiplier breaks the large number into smaller numbers and algorithm called recursively for subpart for performing multiplication. It works on the linear and polynomial function as well. In [6] the author has evaluated Padded, Binary, Simple and Generalized Karatsuba multiplier and proposed a new Hybrid Karatsuba multiplier using Simple and Generalized Karatsuba multiplier. Generalized Karatsuba multiplier is more area efficient compared with other design. In this section, we will discuss the Simple, Generalized and Hybrid Karatsuba multiplier.

The multiplications of two n-bit number perform using three multiplications and add operations. Consider x and y are two n-bit numbers of any base (base-2 or base-10) and the multiplication of these numbers using Karatsuba multiplier are performed using the following formulas. The numbers are divided into Higher and Lower bits. The High bit represent using H and L represents a Lower bit.

$$a= x_H y_H$$
$$d= x_L y_L$$
$$e= (x_H+ x_L)(y_H+y_L)-a-d$$
$$xy=ab^n+eb^{n/2}+d \qquad (5)$$

The above requires only three multiplications and multiplier called recursively until the number being multiplied is a single digit number.

3.1.1 Method for Polynomial Multiplication

The Karatsuba multiplier can also be used for multiplication of polynomials. The finite field multiplication for two polynomial of degree-*n* $A(x)$ *and* $B(x) \epsilon GF(2^n)$ is defined as

$$C(x)=A(x)B(X) \qquad (6)$$

The n-bit multiplicand of equation (6) is divided into two term polynomials and multiplication is perform using three n/2 multiplication which shown below[6].

$$C(x) =(A_hx^{n/2} + A_l)(B_hx^{n/2} + B_l)$$
$$=A_hB_hx^n+(A_hB_l+ A_lB_h)x^{n/2}+A_lB_l$$
$$=A_hB_hx^n+((A_h+A_l)(B_h+B_l)+A_hB_h+A_lB_l)x^{n/2} +A_lB_l$$

3.1.2 Hybrid Karatsuba Multiplier

The Hybrid Karatsuba multiplier[6] is designed using simple and General Karatsuba multiplier which is shown in Figure 5.

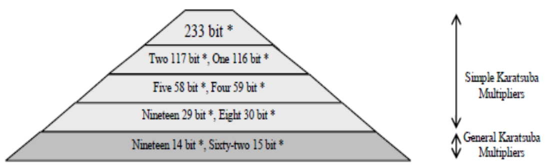

233 bit *
Two 117 bit *, One 116 bit *
Five 58 bit *, Four 59 bit *
Nineteen 29 bit *, Eight 30 bit *
Nineteen 14 bit *, Sixty-two 15 bit *
Simple Karatsuba Multipliers
General Karatsuba Multipliers

Figure 5. Hybrid karatsuba multiplier.

In Hybrid multiplier, the initial multiplication for all large multiplication is done using Simple Karatsuba Multiplier and final small multiplication performs using General Multiplier. The author has implemented 233-bit Hybrid multiplier on FPGA. The result shows that 233-bit Hybrid multiplier is more area efficient, but relatively slower than other Karatsuba design. Let's consider an example of a 4 digit Karatsuba multiplier,

Compute 1234*4321, the subproblems will be,

a1=12*43
d1=34*21
e1=(12+34)*(43+21)–a1–d1 = 46*64–a1–d1

The first subproblem will be,

a1=12*43

This has the following sub problems,

a2=1*4=4
d2=2*3=6
e2=(1+2)(4+3)–a2–d2 =11
Answer: 4*102+11*10+6=516

The Second sub problem is,

d1=34*21

This has the following sub problems,

a2=3*2=6
d2=4*1=4
e2=(3+4)(2+1)–a2–d2 = 11
Answer:6*102+11*10+4=714

The Third sub problem is,

e1=46*64–a1–d1

This has the following sub problems,

a2=4*6=24
d2=6*4=24
e2=(4+6)(6+4)–a2–d2 = 52
Answer:24*102+52*10+24-714 -516 = 1714

And the final answer is,

1234*4321=516*104+1714*100+714
= 5,332,114

This is how Karatsuba multiplier works for large numbers.

3.2 Vedic multiplier

Jagdguru Shakarachraya Bharti Krishna Teerthaji Maharaj proposed different simple methods for all mathematical calculations. Any mathematical calculations perform using Vedic mathematics is simple to implement and faster. The Vedic multiplier is more area and delays efficient than other multipliers[13]. Jagdguru Shakarachraya proposed 16 sutras and 13 sutras for Vedic mathematics from Athrav Veda. Out of this 16 sutras following two sutras are used for multiplication of two numbers.

i. Nikhilam Navatascaramam sutra
ii. Urdhva –Tiryagbhyam sutra

Among this Urdhav-Triyagbhyam sutra is more efficient. In our scalar multiplication, we perform finite field multiplication operation using Urdhav-Triyagbhyam. The Urdhav-Triyagbhyam multiplication technique can be directly applied for decimal and binary number.

3.2.1 Urdhva Tiryagbhyam

Urdhva–Tiryagbhyam sutra is one of the 16 Vedic sutras which perform the multiplication operation of two numbers[15]. The multiplication technique which is used in this sutra is a general technique, which can directly be applied to a decimal, binary, small and large number. The beauty of this sutra is that the same multiplication method can be directly applied to decimal as well as binary numbers. “Urdhva” means vertically and “Tiryagbhyam” means crosswise, therefore, it is also called as Vertically and Crosswise algorithm[15]. Figure 6 shows steps for multiplication of two 3-digit decimal numbers using vertically and crosswise method and Figure 7 shows an alternative method for multiplication of two 4-digit using Urdhva–Tiryagbhyam sutra [4].

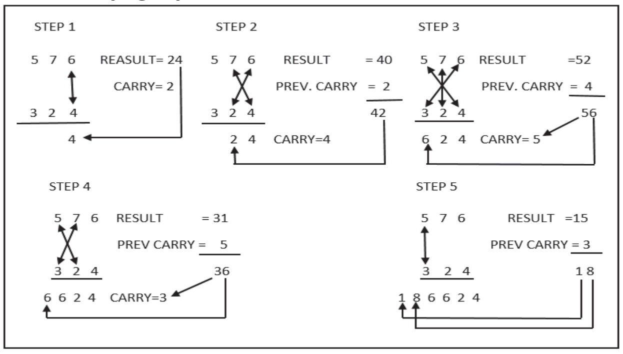


Figure 6. Multiplication of two decimal numbers vertically and crosswise technique.

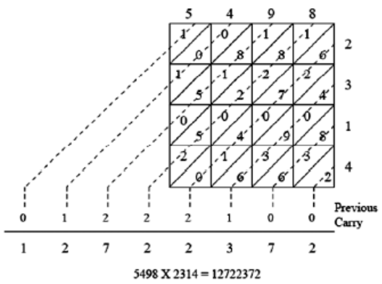


Figure 7. Multiplication of two decimal numbers.

To demonstrate the working of a typical Vedic multiplication algorithm, consider the multiplication of two numbers 42*21[12]. The following steps perform this:

Step1. Multiply the 2 highest digits (4 and 2), which will be resulting in an 8.

Step2. For the next higher digit, cross multiply 4*1 (4) and 2*2 (4), and add together, producing the middle digit of number 8.

Step3. For the lowest digit, multiply the 2 lowest digits (1*2) together, resulting in a 2.

Step4. Put all of the digits together to produce your answer using the Vedic multiplier which is 882

One thing that can and should be noted here is that the order in which you go through for the Vedic process does not actually matter. So, we can similarly start with the lowest digit and work our way up to the highest digit.

3.2.2 Algorithm for 4x4 Vedic Multiplier

The multiplication steps for 4x4 multiplier using vertically and crosswise technique is given below. Consider two 4-digit numbers for multiplication of any base

A= $a_3a_2a_1a_0$

B= $b_3b_2b_1b_0$

Step1: $s_0= a_0*b_0$

Step2: $c_1s_1= a_0*b_1+a_1*b_0$

Step3: $c_2s_2= a_0*b_2+a_1*b_1+a_2*b_0+c_1$

Step4: $c_3s_3= a_0*b_3+a_1*b2+a_2*b_1+ a_3*b_0+c_2$

Step5: $c_4s_4= a_1*b_3+a_2*b_2+a_3*b_1+c_3$

Step6: $c_5s_5= a_2*b_3+a_3*b_2+c_4$

Step7: $c_6s_6= a_3*b_3 + c_5$

Step8: Arrange the digit $c_6s_6s_5s_4s_3s_2s_1s_0$ to get final result.

Once 4x4 multiplier is designed than this multiplier is used recursively to design 8x8, 16x16, 32x32 and higher bit multiplier.

4. FPGA Implementation of Scalar Multiplication

Scalar multiplication involves multiplication of a scalar quantity with a vector quantity, which results in a vector output. This type of multiplication is the most basic operation in the field of vector computation and is used in point multiplication based applications like encryption using ECC. Scalar multiplication usually involves multiple normal multiplications in order to produce the vector result. The following diagram shows the operation of the scalar multiplication.

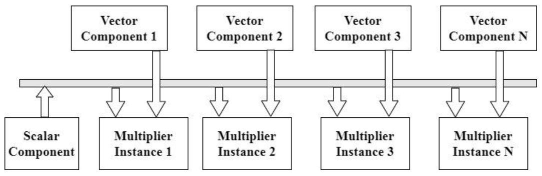


Figure 8. Scalar multiplication using simple multiplier.

From Figure 8, we can see that for an N dimension vector, we need N simple multiplier instances. Thus as the dimension of the vector quantity increases, the number of multipliers increase linearly. If the complexity of a simple multiplier is O(n), then for an N dimension vector, the scalar multiplier complexity will be N*O(n), similarly, the area and power of the scalar multiplier follow the same pattern. Thus it is essential to optimize the simple multiplier unit in order to optimize the performance of the scalar multiplier.

Generally, Karatsuba multiplier is used as the basic building block for the scalar multiplier, the Karatsuba multiplier has many advantages including but not limited to,

- Increased speed of operation when compared to shift and add method
- Less number of computations, thus less area when compared to shift and add method
- Low power consumption

But, the performance of the Karatsuba based scalar multiplier can be further enhanced by using a Vedic multiplier in place of the Karatsuba multiplier. The Vedic multiplier based scalar multiplication diagram can be represented as follows,

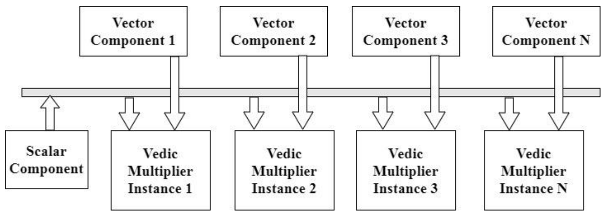

Vector Component 1
Vector Component 2
Vector Component 3
Vector Component N
Scalar Component
Vedic Multiplier Instance 1
Vedic Multiplier Instance 2
Vedic Multiplier Instance 3
Vedic Multiplier Instance N

Figure 9. Scalar multiplication using vedic multiplier

Figure 9 shows; we have replaced the existing normal multiplier with the Vedic multiplier. The Urdhva Tiryakbhyam sutra is used, which is described in the previous section. Using the Vedic multiplier for scalar multiplication design gives the following advantages,

- Delay of the Vedic multiplier is one clock cycle, thus the scalar multiplication happens very quickly
- The power consumption of the circuit reduces as the number of clocks for which the circuit is active is reduced to 1, thereby reducing the overall energy requirement of the system
- Vedic multiplier uses less number of operations when compared to the Karatsuba multiplier, thus the overall area of the scalar multiplier reduces drastically

Figure 10, shows schematic diagram of a 2x2 Vedic multiplier,

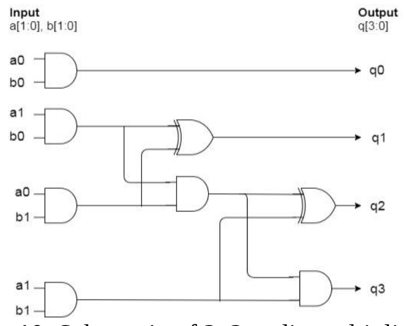


Figure 10. Schematic of 2x2 vedic multiplier using two half adder.

Figure 10, shows the operation of straight and cross as defined by the Urdhva Tiryakbhyam sutra is performed. First the values a0 and b0 are ANDed (straight), then the values a1, b0 and a0, b1 (cross) are ANDed and their respective products are XORed in order to get the sum and carry. Finally a1, b1 (straight) are ANDed and XORed with the previous carry to get the final MSB bit.

The combinations of 4 Vedic multipliers of 2 bits, along with 4-bit adders are sufficient to produce a complete 4-bit multiplier. The block diagram for a 4-bit multiplier can be seen in the following figure,

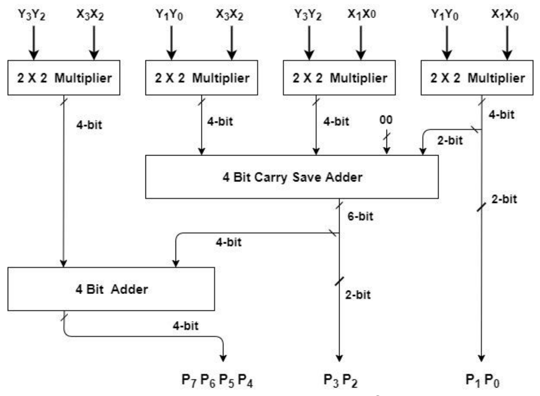


Figure 11. 4-bit vedic multiplier using four 2x2 multipliers.

In Figure 11, the 2x2 multiplier is the same Vedic multiplier, which was previously described. In the 4x4 multiplier, we use the same Urdhva Tiryakbhyam sutra, which first multiplies LSBs of X and Y, then MSB of X with LSB of Y, & LSB of X with MSB of Y, and then finally MSB of X and Y. The final result is shown from the P vector in the above figure. The complete operation doesn't require any recursion (like Karatsuba multiplier), and thus the entire 4 bits get multiplied in a single clock cycle. There by reducing the delay of the system to 1 clock cycle. A similar process is applied for 8x8, 16x16 and NxN Vedic multiplier in order to perform parallel multiplication. Due to simplicity in construction, the power and area requirements of this design are less too. Based on these advantages, we evaluated the performance of the Vedic multiplier based scalar multiplier and obtained some very interesting results which are described in the next section.

5. Implementation Results

This section present implementation results of Scalar multiplication using Karatsuba Multiplier and Vedic multiplier. The scalar multiplier is designed for the binary field for 233-bit $GF(2^{233})$ which is secured curved recommended by National Institute of Standards Technology(NIST) recommended in his Federal Information Standards(FIPS) 186-3[16]. The Curve value of Curve constant b and base point will be taken from the above standard document is as given below[16].

Curve B-233
Curve Constant
b = 066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce42
81fe115f 7d8f90ad
Base Point P(x, y)
Gx = 0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36
f8f8eb73 71fd558b
Gy = 100 6a08a419 03350678 e58528be bf8a0bef
f867a7ca36716f7e 01f81052

The 32-bit key is used in this design, which is *k* in scalar multiplication used as a private key in ECC. The arithmetic and logic unit (ALU) for Scalar multiplier which calls Karatsuba and the Vedic multiplier is designed using Verilog HDL and implemented on Virtex6 FPGA in Xilinx 13.2 ISE. The Synthesis, PAR reports are used to get the device utilization and delay of the design. Table 1, shows the device utilization summary of Karatsuba and Vedic based Scalar multiplier.

Table 1. Device utilization comparison of karatsuba and vedic scalar multiplier.

	Device Utilization Summary			Area reduction
Slice logic utilization	Scalar using Karatsuba multiplier	Scalar using Vedic multiplier	No of devices availab le	
Number of Slice Registers	48	37	948,48 0	22.92 %
Number of Slice LUTs	3016	2761	474,24 0	8.45%
Number of occupied Slices	999	832	118,56 0	16.72%
Number of LUT Flip Flop pairs used	3016	2764		8.36%
Number of bonded IOBs	501	501	1,200	
Average Fanout of Non-Clock Nets	3.33	2.83		

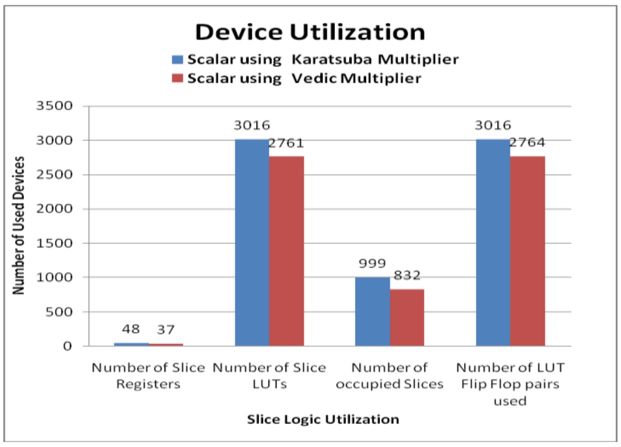


Figure 12. Comparison of device utilization for karatsuba and vedic scalar multiplier.

In Table 2, we have presented the delay comparison of Karatsuba and Vedic based Scalar multiplier. The combinational path delay of the Vedic multiplier is 0.984ns, which is less compared with Karatsuba multiplier 1.117ns. Based on maximum combinational path delay parameter Vedic multiplier based scalar multiplier is 12% more delay efficient compared with Karatsuba multiplier.

Table 2. Delay comparison of karatsuba and vedic scalar multiplier.

Delay parameters	Scalar multiplication using Karatsuba multiplier	Scalar multiplication with Vedic multiplier	Performan ce improvem ent in
Minimum period	1.895ns	0.895ns	52.77%

Minimum input arrival time before the clock	1.438ns	1.267ns	11.89%
Maximum output required time after the clock	2.920ns	1.376ns	52.88%
Maximum combinational path delay	1.117ns	0.984ns	11.91%

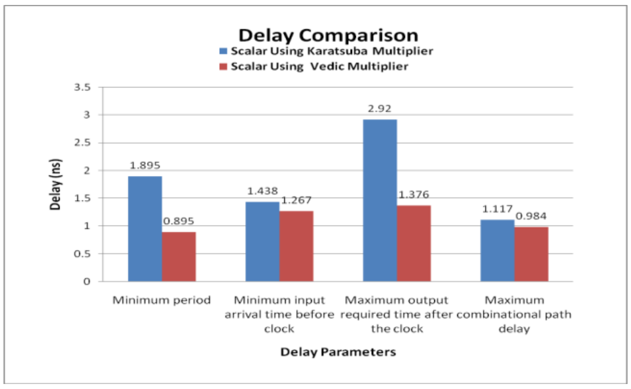
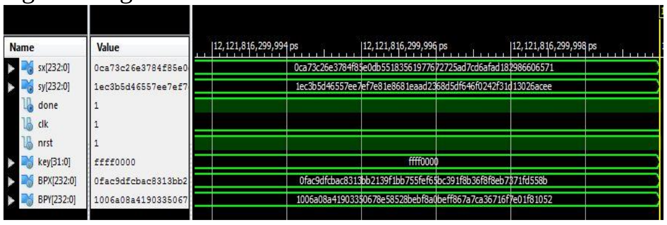


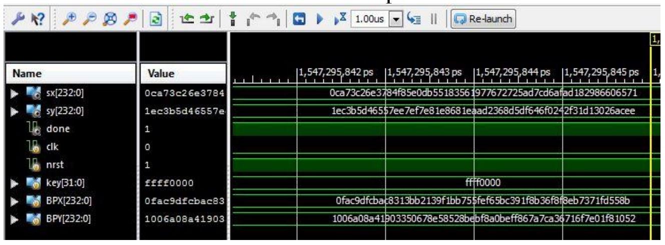
Figure 13. Delay comparison of karatsuba and vedic scalar multiplier.

The test-bench is created for testing the design and design is simulated using ISim simulator. Both the scalar multiplier design are tested with the same data set on Virtex6-xc6vlx760-ff1760 FPGA device. Figure 14 and Figure 15, shows the simulation results of Scalar multiplier design. The base point P(BPX, BPY), key[31:0] is key are the input for the design and Sx, Sy is the resultant values after Scalar multiplication. Initially clock signal is low and when it becomes high the multiplication operation is started and after completion of scalar multiplication, the status of the done signal is high.



```
Name          Value
sx[232:0]     0ca73c26e3784f85e0   0ca73c26e3784f85e0db5518356197767272 5ad7cd6afad182986606571
sy[232:0]     1ec3b5d46557ee7ef7   1ec3b5d46557ee7ef7e81e8681eaad2368d5df646f0242f31d13026acee
done          1
clk           1
nrst          1
key[31:0]     ffff0000             ffff0000
BPX[232:0]    0fac9dfcbac8313bb2   0fac9dfcbac8313bb2139f1bb755fef65bc391f8b36f8f8eb7371fd558b
BPY[232:0]    1006a08a4190335067   1006a08a41903350678e58528bebf8a0beff867a7ca36716f7e01f81052
```

Figure 14. Simulation result of scalar multiplier using karatsuba multiplier.



```
Name          Value
sx[232:0]     0ca73c26e3784   0ca73c26e3784f85e0db5518356197767272 5ad7cd6afad182986606571
sy[232:0]     1ec3b5d46557e   1ec3b5d46557ee7ef7e81e8681eaad2368d5df646f0242f31d13026acee
done          1
clk           0
nrst          1
key[31:0]     ffff0000        ffff0000
BPX[232:0]    0fac9dfcbac83   0fac9dfcbac8313bb2139f1bb755fef65bc391f8b36f8f8eb7371fd558b
BPY[232:0]    1006a08a41903   1006a08a41903350678e58528bebf8a0beff867a7ca36716f7e01f81052
```

Figure 15. Simulation result of scalar multiplier using vedic multiplier.

The Sx and Sy values received after Scalar multiplication is the public key Q used to encrypt the data in Elliptic Curve Cryptography.

6. Conclusions

The proposed work indicates that Vedic multiplier has definitive advantages when compared to Karatsuba multiplier. These advantages are utilized in our paper, and we proposed a scalar multiplier based on Vedic multiplication technique, which outperforms the Karatsuba based multiplier in terms of delay requirement, power consumption, and area requirements. We observe that the Vedic multiplier based implementation is nearly 12% more delay efficient than Karatsuba based implementation, and has 22% less device utilization. Due to which the overall power consumption also reduces. These advantages make the Vedic based scalar multiplication circuit more usable for low power and high speed embedded systems, and also allows for the given circuit to perform better when applied to high complexity applications like encryption and communication. In the future, we plan to integrate the optimized scalar multiplier with a highly complex elliptic curve cryptosystem and analyze its performance.

References

[1] E. Karthikeyan, “Survey of Elliptic Curve Scalar Multiplication Algorithms,” *Int. J. Advanced Networking and Applications*, vol. 04, no. 02, pp. 1581–1590, 2012.

[2] I. Setiadi, A. I. Kistijantoro, and A. Miyaji, “Elliptic curve cryptography: Algorithms and implementation analysis over coordinate systems,” in *International Conference on Advanced Informatics: Concepts, Theory and Applications*, 2015, no. November.

[3] C. Rebeiro and D. Mukhopadhyay, “High Performance Elliptic Curve Crypto-Processor for FPGA Platforms,” *in Proceedings of the 21 st International Conference on VLSI Design , Hyderabad, IEEE Computer Society.*, p. pp 706–711.

[4] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, “Theoretical modeling of elliptic curve scalar multiplier on LUT-based FPGAs for area and speed,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 5, pp. 901–909, 2013.

[5] R. Azarderakhsh and A. Reyhani-masoleh, “Parallel and High-Speed Computations of Elliptic Curve Cryptography Using Hybrid-Double Multipliers,” *IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS*, vol. 26, no. 6, pp. 1668–1677.

[6] C. Rebeiro and D. Mukhopadhyay, “Hybrid Masked Karatsuba Multiplier for GF (2^233),” *in Proceedings of the 11th IEEE VLSI Design and Test Symposium , Kolkata, VLSI Society of India.*, no. 1, p. pp 379-387.

[7] M. Masoumi and H. Mahdizadeh, “Efficient Hardware Implementation of an Elliptic Curve Cryptographic Processor over GF(2^163),” *International Journal of Computer, Electrical, Automation, Control and Information Engineering 2012 International*, vol. 6, no. 5, pp. 725–732, 2012.

[8] W. N. Chelton, S. Member, M. Benaissa, and S. Member, “Fast Elliptic Curve Cryptography on FPGA,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 16, no. 2, pp. 198–205, 2008.

[9] M. M. Panchbhai and U. S. Ghodeswar, “Implementation of Point Addition & Point Doubling for Elliptic Curve,” *IEEE International Conference on Communication and Signal Processing – ICCSP'15*, pp. 746–749, 2015.

[10] B. Ansari, M. A. Hasan, and S. Member, “High-Performance Architecture of Elliptic Curve Scalar Multiplication,” *IEEE TRANSACTIONS ON COMPUTERS, VOL. 57, NO. 11, NOVEMBER 2008*, vol. 57, no. 11, pp. 1443–1453, 2008.

[11] T. T. Nguyen and H. Lee, “Efficient Algorithm and Architecture for Elliptic Curve Cryptographic Processor,” *Journal Of Semiconductor Technology and Science*, vol. 16, no. 1, pp. 118–125, 2016.

[12] V. S. Iyengar, “Novel Elliptic Curve Scalar Multiplication

Algorithms for Faster and Safer Public-Key Cryptosystems," *International Journal on Cryptography and Information Security*, vol. 2, no. 3, pp. 57–66, 2012.

[13] S. Vaidya and D. Dandekar, "Delay-Power Performance Comparison of Multipliers in VLSI," *International Journal of Computer Networks & Communications (IJCNC)*, vol. 2, no. 4, pp. 47–56, 2010.

[14] V. Kaushik and H. Saini, "A Review on Comparative Performance Analysis of Different Digital Multipliers," *Advances in Computational Sciences and Technology*, vol. 10, no. 5, pp. 1257–1272, 2017.

[15] S. P. Pohokar, R. S. Sisal, K. M. Gaikwad, M. M. Patil, and R. Borse, "Design and Implementation of 16 x 16 Multiplier Using Vedic Mathematics," no. Icic, pp. 1174–1177, 2015.

[16] P. National Institute of Standards and Technology Gaithersburg, "Archived publication," *Federal Information Processing Standards Publication, Digital Signature Standard (DSS)*, vol. 3, no. August 2009, 2010.