A Novel Efficient Hardware Implementation of Elliptic Curve Cryptography Scalar Multiplication using Vedic Multiplier

Rakesh K. Kadu

Department of Computer Technology
YCCE, Nagpur, India
rakeshkadu@gmail.com

Dattatraya. S. Adane

Department of Information Technology
RCOEM, Nagpur, India
adaneds@rknec.edu

***Abstract* - We present an integrated circuit 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 is used for public key generation and the performance of ECC greatly depends on it. 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 has 12% less delay than Karatsuba, based scalar multiplier. The scalar multipliers coded in Verilog HDL, synthesize and simulated in Xilinx 13.2 ISE on Virtex6 FPGA.**

***Keywords* - *Elliptic Curve Cryptography, Scalar Multiplication, Karatsuba multiplier, Vedic Multiplier, FPGA*.**

I. INTRODUCTION

Elliptic Curve Cryptography is a 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 provide the same security level compare with the 1024-bit key of RSA. Due to the above feature, this cryptosystem is suitable for devices, having less computation power, limited storage, and limited battery backup. Elliptic curve cryptosystem offers the following protocols 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 above ECC protocols, scalar multiplication will be 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 the literature, many authors have proposed different techniques for optimizing scalar multiplication operation and optimization can be achieved at a different level of computation. The first approach is at the upper level, by representing the [k] in such a way that it reduces the hamming weight of scalar [k]*;* resulting in reducing the execution of addition, 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 achieved at the bottom level by the fast and efficient implementation of underlying finite field operation such as addition, multiplication, squaring and inversion. From above finite field operation, multiplication, inversion is the most time-consuming operation and it occupies more device space.

In [3] the author has 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 parallel multiplier for reducing latency over Edward and generalized Hessian curves. There are many other implementations of scalar multiplication which are presented in [6][7][8][9][10][11][12] using different approaches and methods. In this paper, we propose the finite field multiplication operation using a Vedic multiplier for scalar multiplication. For performance evaluation of 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 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.

II. SCALAR MULTIPLICATION

Scalar multiplication is the most important operations in Elliptic curve cryptography[13]. The scalar multiplication is computing $Q=[k]P$, where k is a scalar and $P(x_1, y1)$ and $Q(x_2, y2)$ are the points on an Elliptic curve E.

The scalar multiplication has the form:

$$Q=[k]*P \quad (1)$$

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

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

The security of ECC depends on the difficulty of Discrete Logarithm Problem (DLP), which is finding k from given P and $Q \in E$. Practically it is very difficult to find k if P and Q are known. Figure 1 shows the layer model of scalar multiplication for computing Q.

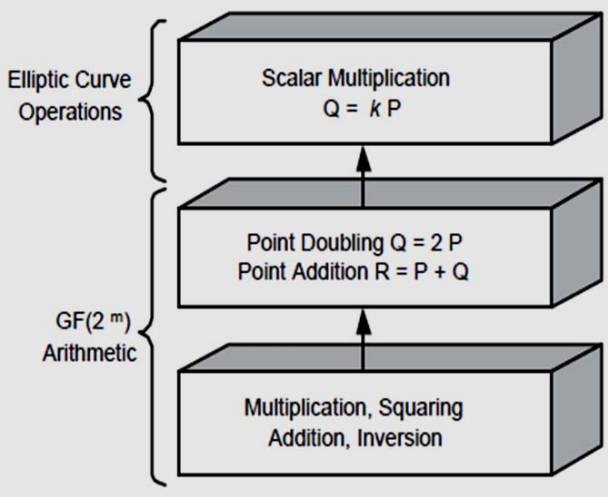

Figure 1. Layer Model for Elliptic Curve Scalar Multiplication

Figure 2 shows the data flow and components of proposed scalar multiplier *ECSM*. The *ECSM* consists of the Control unit *ALU* that calls point addition ECS_{ADD} and point double ECS_{DBL}. ECS_{ADD} and ECS_{DBL} operation computes new coordinate $Q(x, y)$ using finite field arithmetic operations.

The *ECSM* calls ECS_{ADD} and ECS_{DBL} repeatedly based on binary scalar $[k]$ for computing Q. In scalar algorithm scalar $[k]$ is represented in binary representation $k(k_i, \ldots,k_1, k_0)_2$ for $i=n-1$ *to* 0, the multiplication cost depends on the *n-bit* length of $[k]$ and the number of *1's* in $k(k_i, \ldots,k_1, k_0)_2$. In $k(k_i, \ldots,k_1, k_0)_2$, if k_i is 1 then ECS_{ADD} and ECS_{DBL} will perform. If k_i is 0 then we only ECS_{DBL} will be computed. Reducing the hamming weight i.e number of 1's of scalar $[k]$ reduces the number iteration of ECS_{ADD} and ECS_{DBL}, it improves the speed of scalar multiplication.

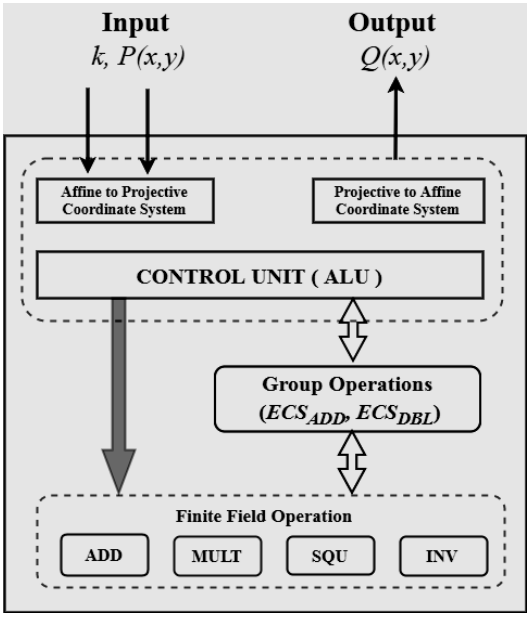

Figure 2. Block Architecture of EC Scalar Multiplication (ECSM).

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: ECC Scalar multiplication using Left to Right Binary Method

Input :
Integer $[k] \neq 0$, $(k_i, \ldots,k_1, k_0)_2$, *where i=n-1 to 0*
Base point $P(x_1,y_1) \in E$.
Output: $Q=[k]P$

```
Begin
Q = O
    for i= n-2  to 0 do
        Q=2Q                    #Point Doubling: ECS_DBL
      if (k_i==1) then
        Q=P+Q                   #Point Addition: ECS_ADD
      end
    end
Return(Q)
end
```

In this paper, we have used the Left to right binary method for scalar multiplication. Algorithm1 shows [1] the steps to compute Q using Left to right binary method which uses point doubling ECS_{DBL} and point addition ECS_{ADD} operation.

For a given point $P(x_1, y_1)$, $Q(x_2, y_2)$ on an Elliptic curve E of $GF(2^m)$, point addition ECS_{ADD} and point doubling ECS_{DBL} of scalar multiplication are computed using (3)(4), resulting in new point $R=(x_3,y_3)$ on the Elliptic curve E in the Affine coordinate system.

$$EC_{ADD}: \text{if } P \neq Q \quad (3)$$
$$x_3=\lambda^2-x_1-x_2$$
$$y_3=\lambda(x_1-x_3)-y_1$$
$$\lambda= y_2-y_1/x_2-x_1$$

$$EC_{DBL}: \text{if } P=Q \quad (4)$$
$$x_3=\lambda^2-2x_1$$
$$y_3=\lambda(x_1-x_3)-y_1$$
$$\lambda = 3x_1^2+a/2y_1$$

If $P \neq Q$ then ECS_{ADD} will perform, and if $P=Q$, then EC_{DBL} operation will be called. The result of ECS_{ADD} or ECS_{DBL} results in a new points R will always be another point on the Elliptic curve E. Figure 3 shows point addition and Figure 4, shows the doubling operation on the elliptic curve E resulting third coordinate $R(x_3, y_3)$ on the same curve E.

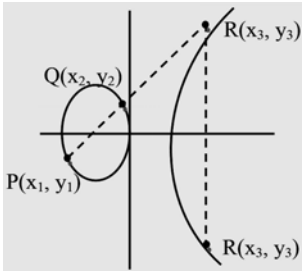

Figure 3. Point Addition: $R=P+Q$.

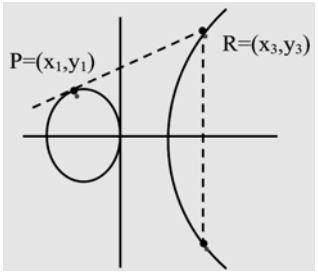

Figure 4. Point Doubling Operation: $R=2P$.

The ECS_{ADD} and ECS_{DBL} use finite field arithmetic operation like addition, subtraction, multiplication, squaring, and inversion to compute coordinate $R(x_3, y_3)$ on the Elliptic Curve E. Since among these finite field operations, multiplication dominates the speed of $ECSM$, we have proposed computation of finite field multiplication using a Vedic multiplier.

III. 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[14], Booth[15], Wallace- Tree[16], Dadda, and Karatsuba[7] are used for performing the multiplication operation. In [7][17] the author has analyzed different multipliers and its variations for their performance. In this section, we will present working of Karatsuba multiplier and a Vedic multiplier.

A. 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 [7] 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 multiplication of two n-bit numbers performs using three multiplications and some addition operations. Consider x and y are two n-bit numbers of any base (base-2 or base-10) and the multiplication of this 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.

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

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

A1. 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:

$C(x)=A(x)B(X)$

The n-bit multiplicand is divided into two term polynomials and multiplication is perform using three n/2 multiplication which shown below[7].

$$C(x) =(A_h x^{n/2} + A_l)(B_h x^{n/2} + B_l)$$
$$=A_h B_h x^n+(A_h B_l+ A_l B_h)x^{n/2}+A_l B_l$$
$$=A_h B_h x^n+((A_h+A_l)(B_h+B_l)+A_h B_h+A_l B_l)x^{n/2} +A_l B_l$$

A2. Hybrid Karatsuba Multiplier

The Hybrid Karatsuba multiplier[7] is designed using simple and General Karatsuba multiplier which is shown in Figure 5.

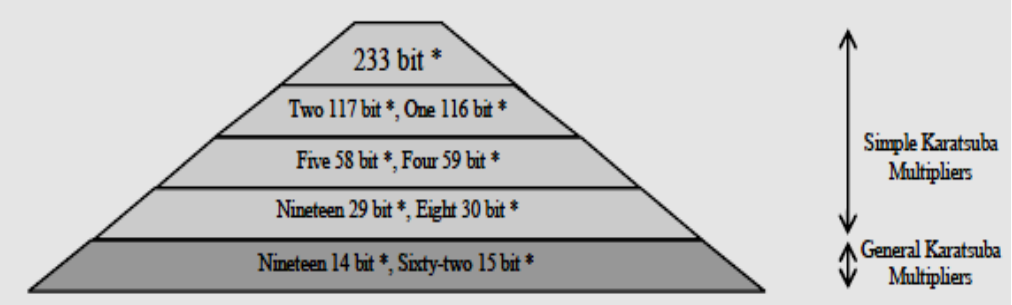

Figure 5. Hybrid Karatsuba Multiplier using Simple and General Karatsuba Multiplier

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 Sub-Problem 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.

B. 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[16]. 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.

B1. Urdhva Tiryagbhyam

Urdhva–Tiryagbhyam sutra is one of the 16 Vedic sutras which perform the multiplication operation of two numbers[18]. The multiplication technique, which is used in this sutra, is a general technique, which can directly be applied to 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[18].

Figure 6 shows steps for multiplication of two 3-digit decimal numbers using vertically and crosswise method and Figure 2 shows an alternative method for multiplication of two 4-digit using Urdhva–Tiryagbhyam sutra [4].

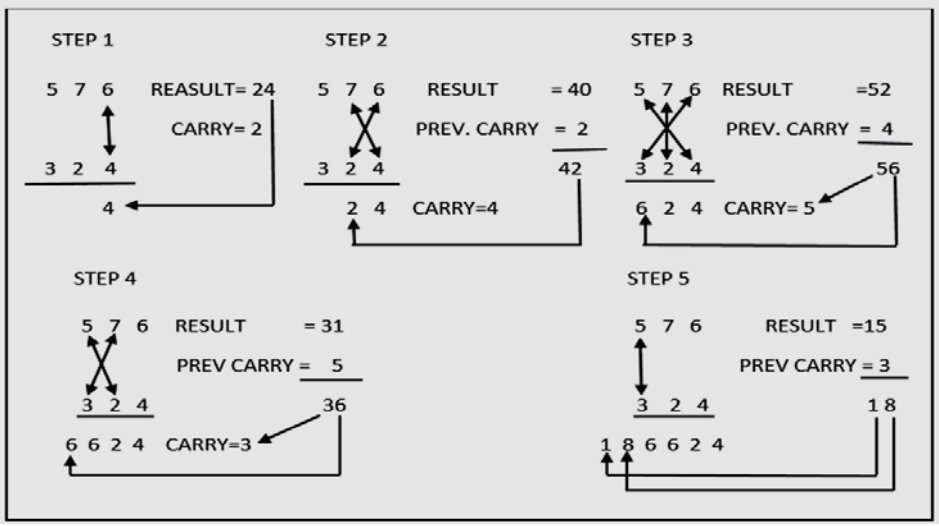

Figure 6. Multiplication of two decimal numbers vertically and crosswise technique [4].

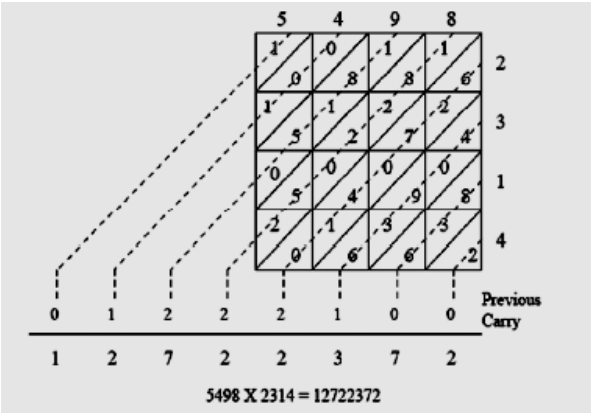

Figure 7. Multiplication of two $(m)_{10}$ x $(n)_{10}$ numbers [4].

To demonstrate the working of a typical Vedic multiplication algorithm, consider the multiplication of two numbers m=42 and n= 21 to obtain o=m*n [12]. The following steps perform this:

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

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

Step3. For the lowest digit, multiply LSB(m) and LSB(n) 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 note that the order in which you go through for the Vedic process does not actually matter. Therefore, we can similarly start with the lowest digit and work our way up to the highest digit.

B2. 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 result.

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

IV. 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 scalar multiplication.

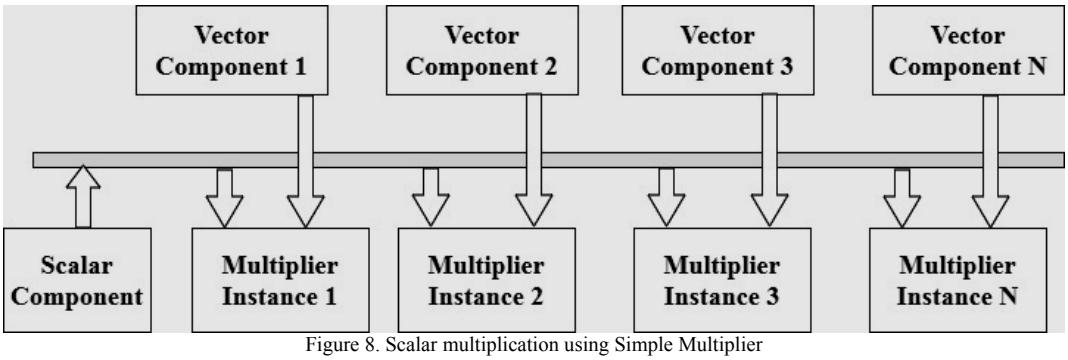

Figure 8. Scalar multiplication using Simple Multiplier

From Figure 8, we can see that for an N dimension vector, we need N simple multiplier instances $(Mui_1, Mui_2, ..., Mui_N)$. 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,

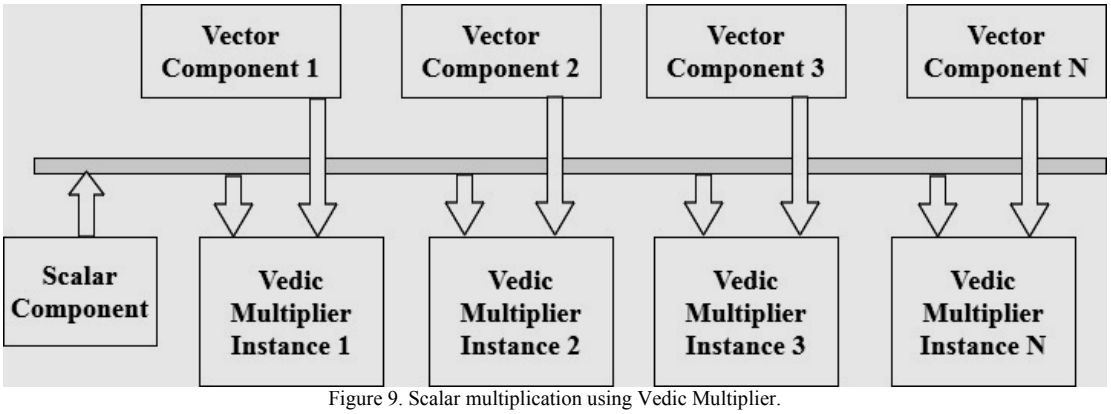

Figure 9. Scalar multiplication using Vedic Multiplier.

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

The block diagram of a 2x2 Vedic multiplier is shown below.

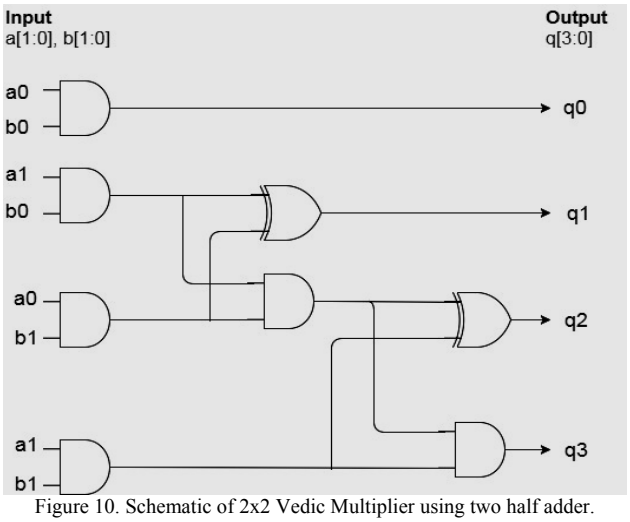

Figure 10. Schematic of 2x2 Vedic Multiplier using two half adder.

Figure 10, shows the operation of straight and cross as defined by the Urdhva Tiryakbhyam sutra is perform. First, the values a_0 and b_0 are ANDed (straight), then the values a_1, b_0 and a_0, b_1 (cross) are ANDed and their respective products are XORed in order to get the sum and carry. Finally, a_1, b_1 (straight) are ANDed and XORed with the previous carry to get the final MSB bit.

The combinations of four 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.

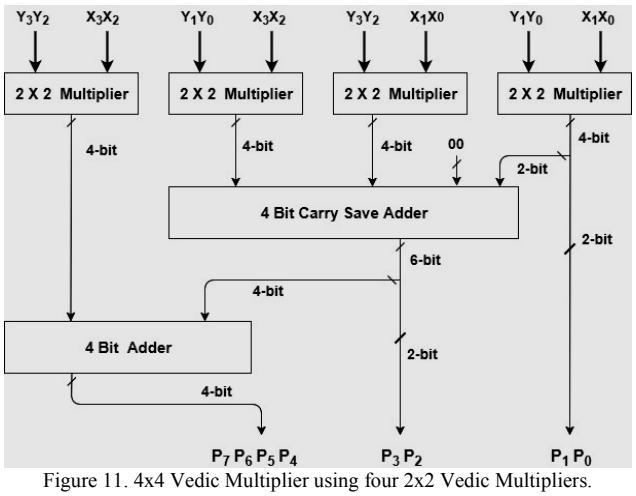

Figure 11. 4x4 Vedic Multiplier using four 2x2 Vedic Multipliers.

In Figure 11, the 2x2 multiplier is the same Vedic multiplier, which is 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 result is shown from the P vector in the above figure. The complete operation does not require any recursion (like Karatsuba multiplier), and thus the entire 4 bits are multiplied in a single clock cycle. Thereby reducing the delay of the system to 1 clock cycle. A similar process will apply 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 that will describe in the next section.

V. Implementation Results

This section present implementation results of *ECSM* Scalar multiplication using Karatsuba multiplier (ECS_{KM}) and Vedic multiplier (ECS_{VM}). 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[19]. The Curve value of Curve constant b and base point will be taken from the above standard document is as given below[19].

Curve: B-233

Curve Constant:

b = 066 647ede6c 332c7f8c 0923bb58 213b333b 20e9ce4281fe115f 7d8f90ad

Base Point P(x,y):

G x = 0fa c9dfcbac 8313bb21 39f1bb75 5fef65bc 391f8b36f8f8eb73 71fd558b

G y = 100 6a08a419 03350678 e58528be bf8a0beff867a7ca36716f7e 01f81052

The *32-bit* key *k,* in scalar multiplication, is a private key in ECC. The Scalar multiplier using Karatsuba ECS_{KM} and the Vedic multiplier ECS_{VM} is coded in Verilog HDL and implemented on Virtex6 FPGA in Xilinx 13.2 ISE. The Synthesis, Place and Route (PAR) report are used to get the device utilization and delay of the design.

Table I summarize the device utilization summary of *233-bit* Karatsuba and Vedic based Scalar multiplier. Slices are the basic building block components in the FPGA fabric. However, each slice contains a number of LUT's (Look-Up-Tables), flip-flops, and carry logic elements which make up the logic of design before mapping. ECS_{VM} occupied 37-slice register, 2761 Slice LUTs, 832 Slices, which is 22.92%, 8.54% 16.72% respectively less, compare to ECS_{KM}. Based on the device utilization summary and obtained data presented in Table I the scalar multiplication using Vedic multiplier consumed fewer devices on FPGA, hence our proposed design is more area efficient then Karatsuba based design.

TABLE I. COMPARISON OF DEVICE UTILIZATION BY KARATSUBA (ECS_{KM}) AND VEDIC SCALAR(ECS_{VM}) MULTIPLIER

Device Utilization Summary obtained from PAR report				
Slice Logic Utilization	**Device utilization by**		**No of Devices Available**	**Space, Area reduction by proposed design**
	ECS_{KM}	ECS_{VM}		
Number of Slice Registers	48	37	948,480	22.92 %
Number of Slice LUTs	3016	2761	474,240	8.45%
Number of occupied Slices	999	832	118,560	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	-	-

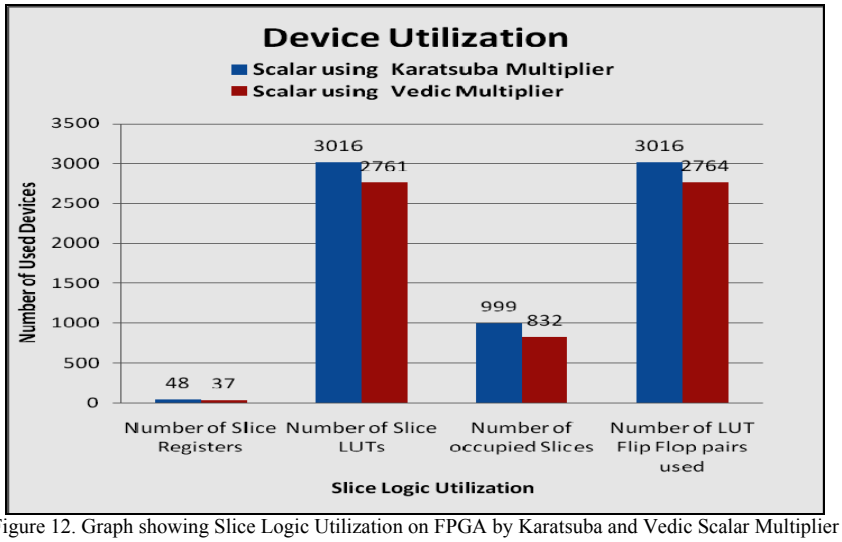

Figure 12. Graph showing Slice Logic Utilization on FPGA by Karatsuba and Vedic Scalar Multiplier

In Table II, we have presented the delay comparison of *233-bit* Karatsuba, Vedic based Scalar multiplier based on timing summary report after synthesis, and Figure13 shows graph plot for the data represented in Table II. The combinational path delay of the ECS_{VM} is *0.984ns*, which is less compare with ECS_{KM} *1.117ns*. Based on maximum combinational path delay there is *12%* speedup using a Vedic multiplier based scalar multiplier then Karatsuba multiplier. Looking at the other delay parameters the input arrival time of ECS_{VM} is *1.267ns* means that ECS_{VM} takes less time to start processing on input data. Similarly, time required to output after clock signal is *1.376ns* which is also less then *2.920ns* of ECS_{KM} and there is *53%* speedup for generating the final output. Based on all the dealy parameters and obtained values we conclude that our proposed Vedic based scalar multiplication ECS_{VM} has more delay efficient then Karatsuba based scalar multiplier ECS_{KM}.

TABLE II. DELAY COMPARISON OF KARATSUBA (ECS_{KM}) AND VEDIC SCALAR(ECS_{VM}) MULTIPLIER

Delay Parameters	ECS_{KM}	ECS_{VM}	**Speedup by the proposed design**
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	2.920ns	1.376ns	52.88%
Maximum combinational path	1.117ns	0.984ns	11.91%

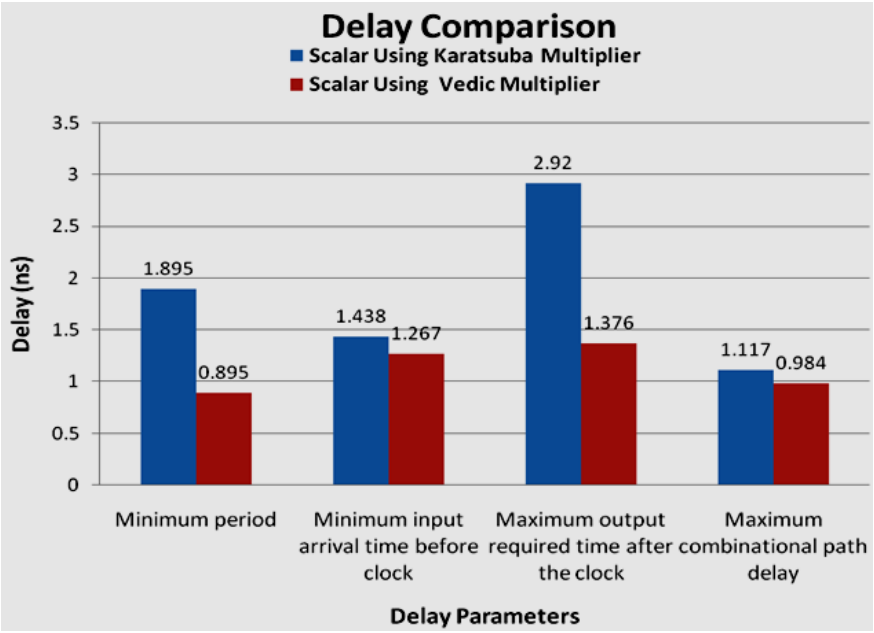

Figure 13. The graph for Delay parameters of Karatsuba and Vedic Scalar Multiplier

The test-bench is created for testing the design and simulated using ISim simulator. ECS_{KM} and ECS_{VM} are tested with the same data set on *Virtex6-xc6vlx760-ff1760* FPGA device. Figure 13 and Figure 14; show 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 i.e. *0* and when it becomes high *1* the multiplication operation is started and after completion of scalar multiplication, the status of the done signal is high to notify regarding completion of scalar multiplication.

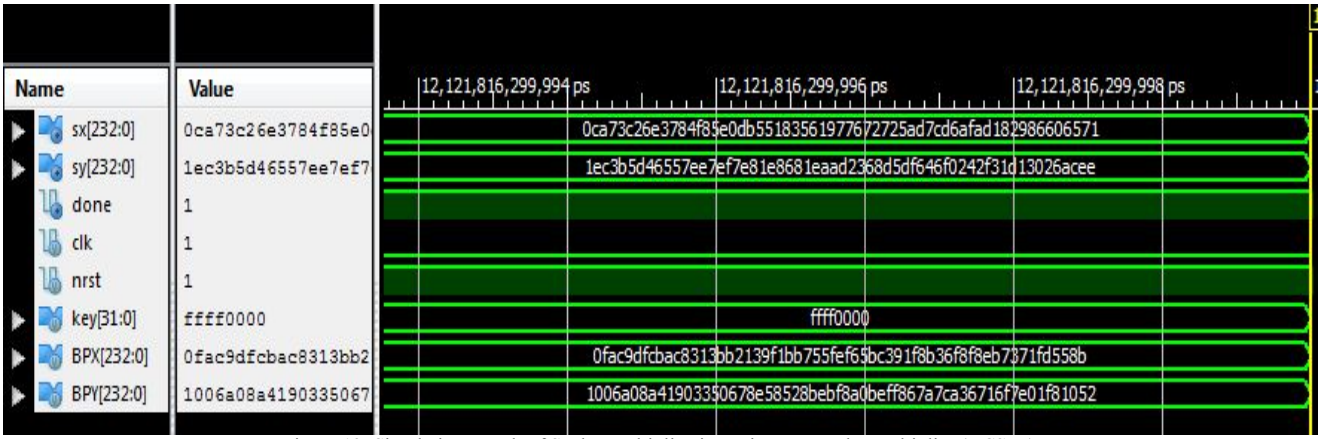

Figure 13. Simulation Result of Scalar Multiplication using Karatsuba Multiplier (ECS_{KM})

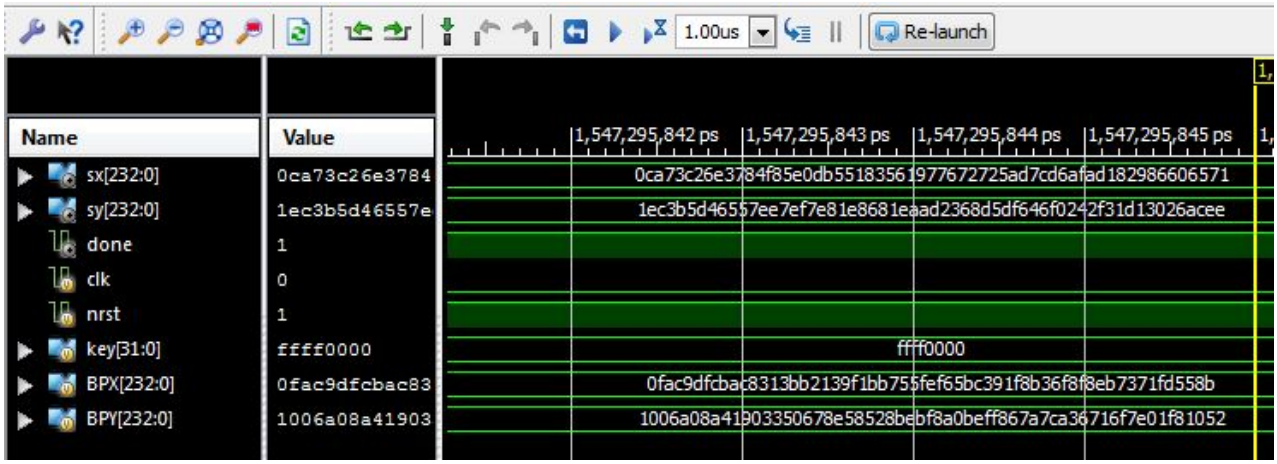

Figure 14. Simulation Result of Scalar Multiplication using Vedic Multiplier (*ECS$_{VM}$*)

The *Q(sx, sy)* are the scalar output received after scalar multiplication of base point *BP(x,y)* and the key value *key(31:0)*. The values received after Scalar multiplication is the public key *Q* used to encrypt the data in Elliptic Curve Cryptography.

VI. Conclusion

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," vol. 1590, no. 02, pp. 1581–1590, 2012.

[2] I. Setiadi, A. Miyaji, and A. I. Kistijantoro, "Elliptic Curve Cryptography : Algorithms and Implementation Analysis over Elliptic Curve Cryptography : Algorithms and Implementation Analysis over Coordinate Systems," no. November 2014.

[3] C. Rebeiro and D. Mukhopadhyay, "HIGH-PERFORMANCE ELLIPTIC CURVE CRYPTO-PROCESSOR FOR FPGA PLATFORMS."

[4] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical Modeling of Elliptic Curve Scalar Multiplier on LUT-Based FPGAs for Area and Speed," 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," vol. 26, no. 6, pp. 1668–1677, 2015.

[6] S. S. Roy, C. Rebeiro, D. Mukhopadhyay, J. Takahashi, and T. Fukunaga, "Scalar Multiplication on Koblitz Curves using," pp. 1–6.

[7] C. Rebeiro and D. Mukhopadhyay, "HYBRID MASKED KARATSUBA MULTIPLIER FOR GF (2 233)," no. 1.

[8] M. Masoumi and H. Mahdizadeh, "Efficient Hardware Implementation of an Elliptic Curve Cryptographic Processor over GF (2 163)," vol. 6, no. 5, pp. 725–732, 2012.

[9] W. N. Chelton, S. Member, M. Benaissa, and S. Member, "Fast Elliptic Curve Cryptography on FPGA," vol. 16, no. 2, pp. 198–205, 2008.

[10] M. M. Panchbhai and U. S. Ghodeswar, "Implementation of Point Addition & Point Doubling for Elliptic Curve," pp. 746–749, 2015.

[11] B. Ansari, M. A. Hasan, and S. Member, "High-Performance Architecture of Elliptic Curve Scalar Multiplication," vol. 57, no. 11, pp. 1443–1453, 2008.

[12] T. T. Nguyen and H. Lee, "Efficient Algorithm and Architecture for Elliptic Curve Cryptographic Processor," vol. 16, no. 1, pp. 118–125, 2016.

[13] V. S. Iyengar, "NOVEL E ELLIPTIC CURVE SCALAR MULTIPLICATION ALGORITHMS FOR FASTER AND SAFER PUBLIC -KEY," vol. 2, no. 3, pp. 57–66, 2012.

[14] K. S. Gurumurthy and M. S. Prahalad, "Fast and Power Efficient 16 × 16 Array of Array Multiplier using Vedic Multiplication," vol. 110, pp. 1–4.

[15] "TASK 1 : 8-bit Verilog Code for Booth ' s Multiplier Testbench for Booth ' s Multiplier."

[16] S. Vaidya and D. Dandekar, "DELAY-POWER PERFORMANCE COMPARISON OF MULTIPLIERS IN VLSI," vol. 2, no. 4, pp. 47–56, 2010.

[17] V. Kaushik and H. Saini, "A Review on Comparative Performance Analysis of Different Digital Multipliers," vol. 10, no. 5, pp. 1257–1272, 2017.

[18] 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.

[19] F. Publication, "Archived publication," vol. 3, no. June 2009, 2013.