MODELLING OF ELLIPTIC CURVE SCALAR MULTIPLIER ON LUT-BASED FPGAS FOR AREA AND SPEED

Sakthi shree.E[1] ,
Student,Dept. Of ECE
Sathyabama University,
Chennai,Tamilnadu,India.
esakthiettikkan@gmail.com

Jagatheswari.S[2]
Student,Dept. Of ECE
Sathyabama University
Chennai,Tamilnadu,India

Mrs.P.Kavipriya[3]
Assistant professor,Dept.Of ECE
Sathyabama University,
Chennai,Tamilnadu,India
ecekavipriya@gmail.com

Abstract-The aim of this paper is to design a karatsuba multiplier that reduces the area consumed by modifying the algorithms Montgomery ladder and Itoh-tsujii algorithm. Here a theoretical model to approximate the delay of different characteristic two primitives used in an elliptic curve scalar multiplier architecture (ECSMA) implemented on k input lookup table (LUT)-based field-programmable gate arrays. Approximations are used to determine the delay of the critical paths in the ECSMA. This is then used to theoretically estimate the optimal number of pipeline stages and the ideal placement of each stage in the ECSMA. This paper illustrates suitable scheduling for performing point addition and doubling in a pipelined data path of the ECSMA. Finally, detailed analyses,supported with experimental results, are provided to design the fastest scalar multiplier over generic curves. Experimental results for GF(2163) show that, when the ECSMA is suitably pipelined,the scalar multiplication can be performed in only 13.90 ns on a Xilinx Virtex V.

Keywords-ECSMA, Karatsuba multiplier, Montgomery ladder,Itoh-tsujii algorithm

I INTRODUCTION

The rapid advances in information technology in the past few decades have led to intensive research on information security. Many technologies and cryptographical systems have been developed, all to secure information and protect it from unauthorized invaders. Public-key cryptography has been widely studied and used since 1975 when Rivest, Shamir, and Adleman invented RSA public key cryptography. This system heavily depends on integer factorization problem (IFP) using big key bits such as 1024 bits and 2048 bits. Later on Deffie-Hellman in developed the public key exchange algorithm using the discrete logarithm problem (DLP). ElGamal also used DLP in encryption and digital signature scheme. In 1985, Koblitz and Miller independently used EC in cryptography using elliptic curves discrete logarithm problem (ECDLP). In recent years, researchers have given more attention to develop the proposed ECC algorithms and improve their efficiency. Improving the efficiency of scalar multiplication in EC is one of the main interests of many researchers in the field of cryptology.

The techniques proposed so far use different methods for representing the scalar *k*, which clearly shows different levels of computation speed and security. Binary representation is extended to signed binary representation, and its Non-Adjacent Form (NAF) algorithm . Other well-known techniques such as the window methods and the Montgomery method have brought about much improvement in terms of the efficiency of EC arithmetic. When doubling one point *P* to obtain 2*P* = *R* as a new point on E, extra field squaring over prime fields is required, but it is the same cost as in point adding if the curve has been defined over the binary fields. Adding two points *P* and *Q* on the same elliptic curve *E*, requires solving three equations, that involving one field inversion, one field squaring, and two field multiplications, which are costly operations in EC implementation. Some other operations involved in adding two points are addition, subtraction, and multiplication of small integers, which are in most cases neglectible operations. The proposed algorithms offers many different formulas for finding point multiplication from the given point *P*. One can use many doublings 2(...(2(2*P*))) with one or more extra point additions. The reverse operation also lead to find points such as P from (2*P*), which is called *point halving.*

Elliptic curve cryptography is rapidly becoming the standard for public-key ciphers because of the large amount of security provided per key bit. Several accreditation bodies have migrated to ECC for their public-key cryptographic requirements. To match the speed requirements for real-time applications, hardware acceleration of ECC is a necessity. Field-programmable gate arrays Forman ideal platform for hardware implementations of security algorithms such as ECC. However, because FPGAs are resource-constrained, there are many challenges in developing designs to ensure better resource utilization, keeping the critical delay minimal. In this paper, we present the design of a high-speed ECC processor for binary fields on FPGA platforms. This paper proposes an efficient implementation of the field multiplier, which is the most important primitive in ECC. The elliptic curve scalar multiplier architecture presented uses a pipelined bit-parallel Karatsuba multiplier, with the objective of improving the utilization of the FPGA's lookup tables. For the inversion, another important field primitive, the Itoh–Tsujii algorithm, is designed to use optimal exponentiation circuits specific to the LUT size of the underlying FPGA platform. These field primitives are combined to realize the elliptic curve scalar multiplier. In the next part, this paper explores opportunities for pipelining the design for high-speed.

II KARATSUBA MULTIPLIER

Basically Karatsuba stated that if we have to multiply two n-digit numbers x and y, this can be done with the following operations, assuming that B is the base of and m < n.

First both numbers x and y can be represented as x1,x2 and y1,y2 with the following formula.

$x=x1 * B^m + x2$; $y= y1 * B^m + y2$ ----(1)

Obviously now xy will become as the following product.

$Xy=(x1 * B^m + x2)(y1 * B^m +y2) =>$

$a= x1 * y1$;

$b = x1 * y2 + x2 * y1$;

$c = x2 * y2$ -----(2)

Finally xy will become:

$xy = a * B^{2m} + b * B^m + c$

However a, b and c can be computed at least with four multiplication, which isn't a big optimization. That is why Karatsuba came up with the brilliant idea to calculate b with the following formula:

$b = (x1 + x2)(y1 + y2) - a – c$ ----(3)

That make use of only three multiplications to get xy.

Let's see this formula by example.

47 x 78
x = 47
x = 4 * 10 + 7
x1 = 4
x2 = 7
y = 78
y = 7 * 10 + 8
y1 = 7
y2 = 8
a = x1 * y1 = 4 * 7 = 28
c = x2 * y2 = 7 * 8 = 56
b = (x1 + x2)(y1 + y2) - a - c = 11 * 15 - 28 - 56

Now the thing is that 11 * 15 it's again a multiplication between 2-digit numbers,but fortunately we can apply the same rules two them. This makes the algorithm of Karatsuba a perfect example of the "divide and conquer" algorithm.

A.Efficiency analysis:

Karatsuba's basic step works for any base *B* and any *m*, but the recursive algorithm is most efficient when *m* is equal to *n*/2, rounded up. In particular, if *n* is 2^k, for some integer *k*, and the recursion stops only when *n* is 1, then the number of single-digit multiplications is 3^k, which is n^c where $c = \log_2 3$.

Since one can extend any inputs with zero digits until their length is a power of two, it follows that the number of elementary multiplications, for any *n*, is at most $3^{\lceil \log_2 n \rceil} \leq 3n^{\log_2 3.}$

Since the additions, subtractions, and digit shifts (multiplications by powers of *B*) in Karatsuba's basic step take time proportional to *n*, their cost becomes negligible as *n* increases. More precisely, if *t*(*n*) denotes the total number of elementary operations that the algorithm performs when multiplying two *n*-digit numbers, then

$t(n)=3t(\lceil n/2 \rceil) + cn +d$ ----(4)

for some constants *c* and *d*. For this recurrence relation, the master theorem gives the asymptotic bound $t(n) = \Theta(n^{\log_2 3})$.

It follows that, for sufficiently large *n*, Karatsuba's algorithm will perform fewer shifts and single-digit additions than longhand multiplication, even though its basic step uses more additions and shifts than the straightforward formula. For small values of *n*, however, the extra shift and add operations may make it run slower than the longhand method. The point of positive return depends on the computer platform and context. As a rule of thumb, Karatsuba is usually faster when the multiplicands are longer than 320–640 bits.

III IMPLEMENTATION OF FIELD PRIMITIVES

In polynomial basis representation of binary fields, addition is performed by bitwise EXCLUSIVE OR operation. For fixed irreducible polynomials, squaring circuits can be hardwired, thus making squaring operations easy. Operations such as multiplications and inversions are complex and should be implemented efficiently. This section describes the implementation of the field multiplication and inversion units in the ECSMA.

A.Field multiplication

Multiplication in binary fields involves a polynomial multiplication followed by a modular reduction. For high-speed implementations, bit-parallel multipliers are preferred to serial and word-parallel multipliers. Several algorithms for bit parallel field multiplication are available in the literature. The ECSMA in this paper uses a Karatsuba multiplier because of its sub quadratic complexity. It is shown in fig 3.1.

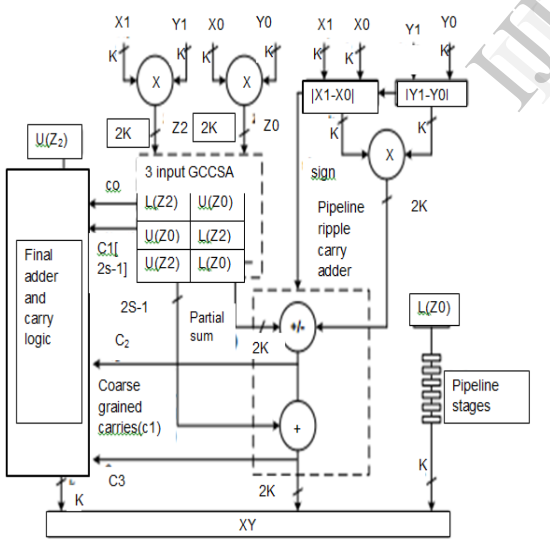


Fig :3.1: Block diagram of recursive Karatsuba multiplier

IV PARAMETRIC KARATSUBA INTEGER MULTIPLIER

The key to our Montgomery multiplier design is the recursive Karatsuba algorithm. This allows us to compute modular multiplication with a complexity approaching. Our design uses multiple-precision arithmetic techniques so that the critical path delay is independent of the multiplier’s bit-width. Unless stated otherwise, we assume we are multiplying two 2k-bit unsigned integers and the limb-widths of all components are w. The number of limbs in a 2k-bit word. We use either a coarse-grained carry-save technique or a pipelined multiple-precision technique in all of our adders and subtracters. The critical path of the circuit primarily depends upon the limb-width.

V PROCESSOR ORGANIZATION

In this section, we describe the construction of the ECSMA, which uses the left-to-right double and add algorithm with binary signed digit representation of the scalar Algorithm 1. The scalar s is the input to the processor and the output is the scalar product sP. Point arithmetic is done in the LD projective coordinate system. At every clock cycle, the register bank feeds the arithmetic unit through six buses. At the end of the clock cycle, the results of the computation are stored in the registers through buses C0, C1, C2, and Qout. Control signals are generated at every clock depending on the state of the machine and key digit. This section elaborates the working of the ECSMA shown in Fig 5.1

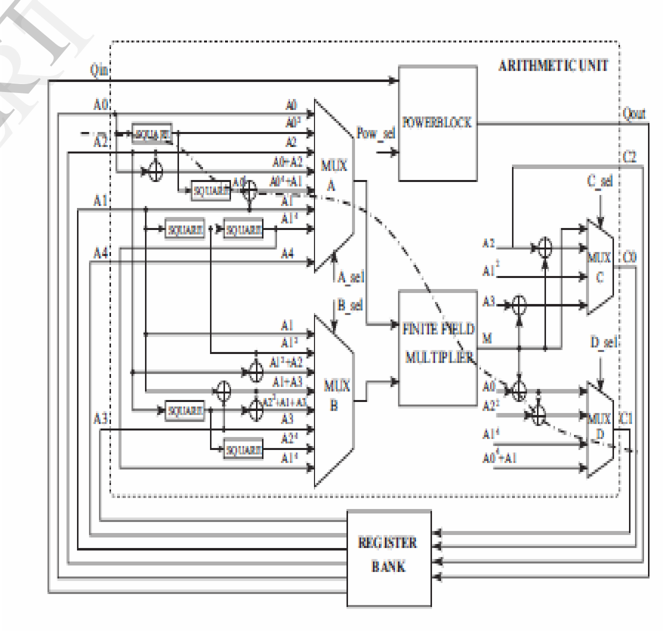


Fig 5.1: ECSMA

VI PIPELINING THE ECSMA

The ECSMA discussed in the previous section suffers from low operating frequency because of the long combinational paths. The operating frequency can be increased by splitting the critical paths using the pipeline strategy. Data dependencies present in the point arithmetic steps may introduce bubbles in the pipeline, resulting in an increase in the clock cycle requirement. The number of bubbles during execution is likely to increase with the number of pipeline stages. Thus, with the increase in pipeline stages, though the delay reduces, the clock cycle requirement increases. Therefore designing a pipeline would require a balance between clock cycle requirement and critical delay. This section first identifies the critical paths in the ECSMA and then theoretically estimates the best pipeline strategy for the design.

VII ALGORITHMS

A.Itoh-Tsujii algorithm

The Itoh-Tsujii inversion algorithm is used to invert elements in a finite field. It was introduced in 1988 and first used over GF(2^m) using the normal basis representation of elements, however the algorithm is generic and can be used for other bases, such as the polynomial basis. It can also be used in any finite field, GF(p^m).

The algorithm is as follows:

```
Input: $A \in GF(p^m)$
Output: $A{-}1$

1.  $r \leftarrow (p^m - 1)/(p - 1)$
2.  compute $A^{r-1}$ in $GF(p^m)$
3.  compute $A^r = A^{r-1} \cdot A$
4.  compute $(A^r)^{-1}$ in $GF(p)$
5.  compute $A^{-1} = (A^r)^{-1} \cdot A^{r-1}$
6.  return $A^{-1}$
```

This algorithm is fast because steps 3 and 5 both involve operations in the subfield GF(p). Similarly, if a small value of p is used a lookup table can be used for inversion in step 4. The majority of time spent in this algorithm is in step 2, the first exponentiation. This is one reason why this algorithm is well-suited for the normal basis, since squaring and exponentiation are relatively easy in that basis.

B .Montgomery ladder

Points on an elliptic curve E, defined over a finite field GF (q), along with a special point called infinity, and a group operation known as point additio n, form a commutative finite group. If P is a point on the curve E, and k is a positive integer computing(eqn 5)

$$kP = P + P + P + \cdot \cdot \cdot + P \qquad \text{----(5)}$$

is called scalar multiplication. The result of scalar multiplication is another point Q on the curve E. It is normally expressed as Q = kP . If E is an elliptic curve defined over GF (q), the number of points in E(GF (q)) is called the order of E over GF (q), denoted by #E(GF (q)). For cryptographic applications #E(GF (q)) = rh where r is prime and h is a small integer and P and Q have order r. Scalars such as k are random integers where $1 < k < r - 1$. Since $r \approx q$, the binary representation of $k = \quad k_i 2^i$ has n bits where $n \approx m = \lceil \log_2 q \rceil$. Scalar multiplication is the most dominant computation part of elliptic curve cryptography.

Algorithm A shows the Montgomery scalar multiplication scheme for non-supersingular elliptic curves over binary fields as it was introduced . In this algorithm Madd(X_1,Z_1,X_2,X_2), Mdouble(X_1,Z_1) and Mxy(X_1,Z_1,X_2,X_2) are functions for point addition, point doubling and conversion of projective coordinates to affine coordinates.

C. Point multiplication technique

Point Multiplication is the basic computation primitive of elliptic curve cryptography. The definition of corresponding operations depends on a particular field, but they always amount to combinations of arithmetic operation.

$$kP = P + P + P + \cdot \cdot \cdot + P \qquad \text{----(6)}$$

Where P is a point on an elliptic curve E and k is an integer in a range $1 \le k <$ order (P). Accordingly, the elliptic curve point multiplication means that the point P is added to itself k times. The order of the point P is n0 if and only if P multiplied with n0 results in the point at infinity.

D. Elliptic curve point addition & doubling

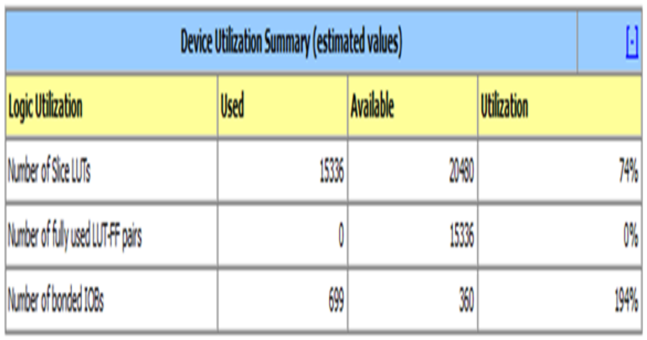
Point addition:
To add two distinct points P and Q on an elliptic curve.
Point doubling:
The point-doubling operation amounts to squaring operations of any binary number.

E..Finite field arithmetic

A field F (finite field) is equipped with two operations, addition and multiplication. Subtraction of field elements is defined in terms of addition.

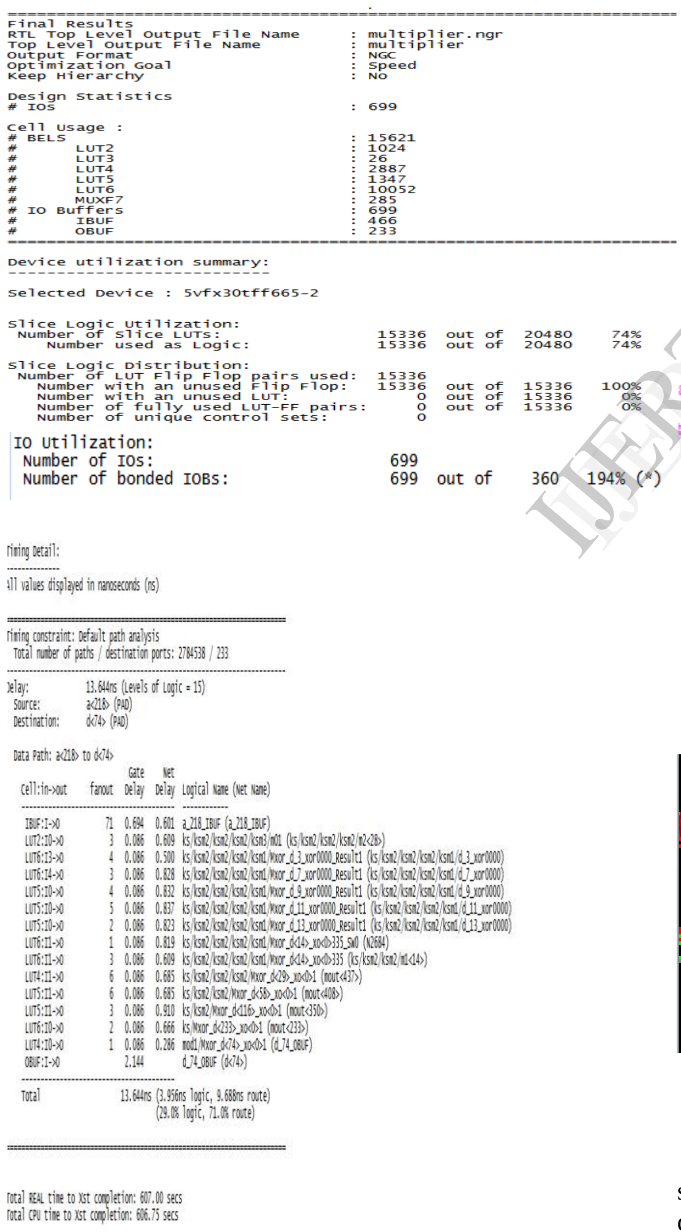
VIII RESULTS AND DISCUSSIONS

The simulation results of Karatsuba Multiplier is shown below. The result in figure 8.1 and figure 8.2 shows the area utilized in the existing system.



Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	15336	20480	74%
Number of fully used LUT-FF pairs	0	15336	0%
Number of bonded IOBs	699	360	194%

Fig 8.1 Device Utilization summary of existing system



```
=========================================================================
Final Results
RTL Top Level Output File Name      : multiplier.ngr
Top Level Output File Name          : multiplier
Output Format                       : NGC
Optimization Goal                   : Speed
Keep Hierarchy                      : No

Design Statistics
# IOs                               : 699

Cell Usage :
# BELS                              : 15621
#      LUT2                         : 1024
#      LUT3                         : 26
#      LUT4                         : 2887
#      LUT5                         : 1347
#      LUT6                         : 10052
#      MUXF7                        : 285
# IO Buffers                        : 699
#      IBUF                         : 466
#      OBUF                         : 233
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 5vfx30tff665-2


Slice Logic Utilization:
 Number of Slice LUTs:                15336  out of  20480    74%
    Number used as Logic:             15336  out of  20480    74%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:  15336
   Number with an unused Flip Flop:   15336  out of  15336   100%
   Number with an unused LUT:             0  out of  15336     0%
   Number of fully used LUT-FF pairs:     0  out of  15336     0%
   Number of unique control sets:         0
IO Utilization:
 Number of IOs:                     699
 Number of bonded IOBs:             699  out of    360   194% (*)


Timing Detail:
--------------
All values displayed in nanoseconds (ns)

=========================================================================
Timing constraint: Default path analysis
  Total number of paths / destination ports: 2784538 / 233
-------------------------------------------------------------------------
Delay:               13.644ns (Levels of Logic = 15)
  Source:            a<218> (PAD)
  Destination:       d<74> (PAD)

  Data Path: a<218> to d<74>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            71   0.694   0.601  a_218_IBUF (a_218_IBUF)
     LUT2:I0->O            3   0.086   0.609  ks/ksm2/ksm2/ksm2/ksm3/m01 (ks/ksm2/ksm2/ksm2/m2<28>)
     LUT6:I3->O            4   0.086   0.500  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d_3_xor0000_Result1 (ks/ksm2/ksm2/ksm2/ksm1/d_3_xor0000)
     LUT6:I4->O            3   0.086   0.828  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d_7_xor0000_Result1 (ks/ksm2/ksm2/ksm2/ksm1/d_7_xor0000)
     LUT5:I0->O            4   0.086   0.832  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d_9_xor0000_Result1 (ks/ksm2/ksm2/ksm2/ksm1/d_9_xor0000)
     LUT5:I0->O            5   0.086   0.837  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d_11_xor0000_Result1 (ks/ksm2/ksm2/ksm2/ksm1/d_11_xor0000)
     LUT5:I0->O            2   0.086   0.823  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d_13_xor0000_Result1 (ks/ksm2/ksm2/ksm2/ksm1/d_13_xor0000)
     LUT6:I1->O            1   0.086   0.819  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d<14>_xo<0>335_SW0 (N2684)
     LUT6:I1->O            3   0.086   0.609  ks/ksm2/ksm2/ksm2/ksm1/Mxor_d<14>_xo<0>335 (ks/ksm2/ksm2/m1<14>)
     LUT4:I1->O            6   0.086   0.685  ks/ksm2/ksm2/ksm2/Mxor_d<29>_xo<0>1 (mout<437>)
     LUT5:I1->O            6   0.086   0.685  ks/ksm2/ksm2/Mxor_d<58>_xo<0>1 (mout<408>)
     LUT5:I1->O            3   0.086   0.910  ks/ksm2/Mxor_d<116>_xo<0>1 (mout<350>)
     LUT6:I0->O            2   0.086   0.666  ks/Mxor_d<233>_xo<0>1 (mout<233>)
     LUT4:I0->O            1   0.086   0.286  mod1/Mxor_d<74>_xo<0>1 (d_74_OBUF)
     OBUF:I->O                 2.144          d_74_OBUF (d<74>)
    ----------------------------------------
    Total                     13.644ns (3.956ns logic, 9.688ns route)
                                       (29.0% logic, 71.0% route)

=========================================================================


Total REAL time to Xst completion: 607.00 secs
Total CPU time to Xst completion: 606.75 secs
```

Fig 8.2 Simulation results of existing system

Figure 8.3 shows the top level module implementation (RTL Schematic) of the Multiplier.

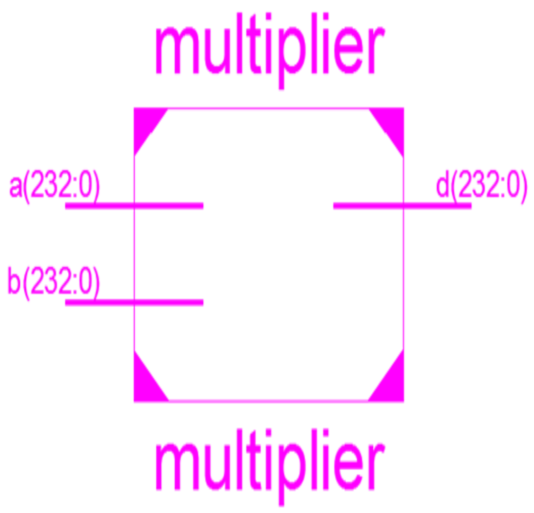


Fig 8.3 RTL Schematic of multiplier

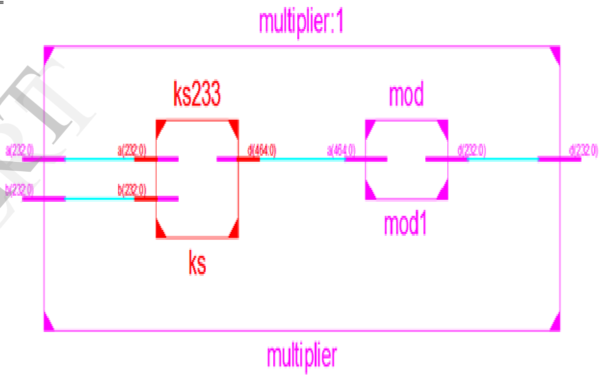


Fig 8.4 Internal RTL Schematic of Multiplier

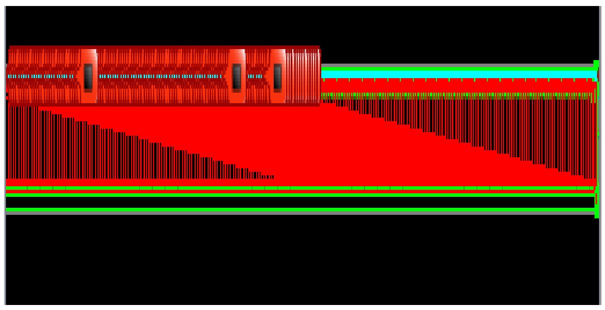
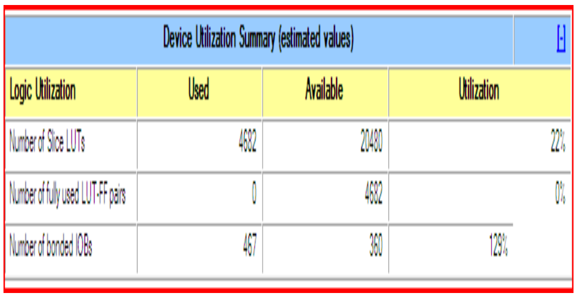


Fig 8.5 Internal Schematic of ks233

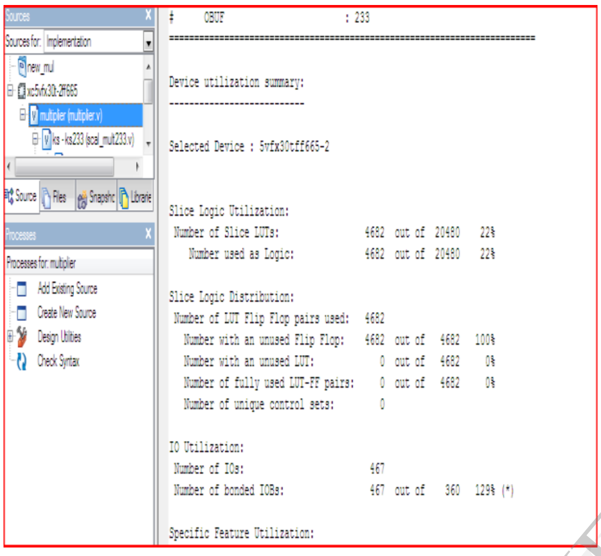
Figure 8.6 shows the device utilization summary of proposed system. Figure 8.7 shows the detailed analysis of the proposed system.

5

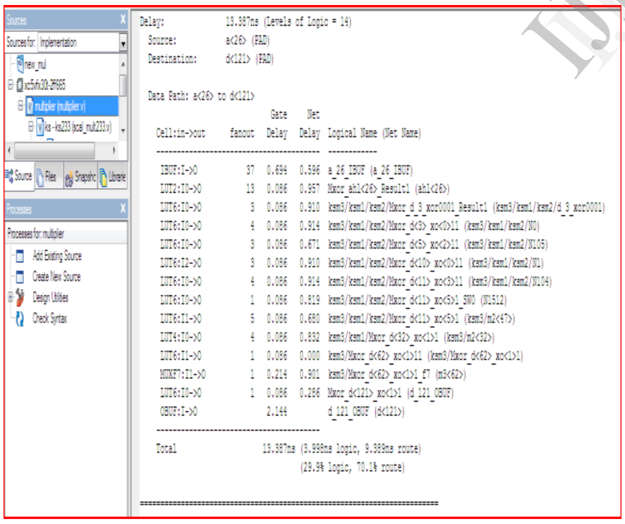


Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice LUTs	4682	20480	22%
Number of fully used LUT-FF pairs	0	4682	0%
Number of bonded IOBs	467	360	129%

Fig 8.6 Device Utilization summary of proposed system



```
#      OBUF                        : 233
=========================================================================


Device utilization summary:
---------------------------


Selected Device : 5vfx30tff665-2



Slice Logic Utilization:
 Number of Slice LUTs:                  4682  out of  20480    22%
    Number used as Logic:               4682  out of  20480    22%

Slice Logic Distribution:
 Number of LUT Flip Flop pairs used:    4682
   Number with an unused Flip Flop:     4682  out of   4682   100%
   Number with an unused LUT:              0  out of   4682     0%
   Number of fully used LUT-FF pairs:      0  out of   4682     0%
   Number of unique control sets:          0

IO Utilization:
 Number of IOs:                          467
 Number of bonded IOBs:                  467  out of    360   129% (*)

Specific Feature Utilization:
```



```
Delay:               13.387ns (Levels of Logic = 14)
  Source:            a<26> (PAD)
  Destination:       d<121> (PAD)

  Data Path: a<26> to d<121>
                                Gate     Net
    Cell:in->out      fanout   Delay   Delay  Logical Name (Net Name)
    ----------------------------------------  ------------
     IBUF:I->O            37   0.694   0.596  a_26_IBUF (a_26_IBUF)
     LUT2:I0->O           13   0.086   0.957  Mxor_ahl<26>_Result1 (ahl<26>)
     LUT6:I0->O            3   0.086   0.910  ksm3/ksm1/ksm2/Mxor_d_3_xor0001_Result1 (ksm3/ksm1/ksm2/d_3_xor0001)
     LUT6:I0->O            4   0.086   0.914  ksm3/ksm1/ksm2/Mxor_d<3>_xo<0>11 (ksm3/ksm1/ksm2/N0)
     LUT6:I0->O            3   0.086   0.671  ksm3/ksm1/ksm2/Mxor_d<5>_xo<2>11 (ksm3/ksm1/ksm2/N105)
     LUT6:I2->O            3   0.086   0.910  ksm3/ksm1/ksm2/Mxor_d<10>_xo<0>11 (ksm3/ksm1/ksm2/N1)
     LUT6:I0->O            4   0.086   0.914  ksm3/ksm1/ksm2/Mxor_d<11>_xo<3>11 (ksm3/ksm1/ksm2/N104)
     LUT6:I0->O            1   0.086   0.819  ksm3/ksm1/ksm2/Mxor_d<11>_xo<5>1_SW0 (N1512)
     LUT6:I1->O            5   0.086   0.680  ksm3/ksm1/ksm2/Mxor_d<11>_xo<5>1 (ksm3/m2<47>)
     LUT4:I0->O            4   0.086   0.832  ksm3/ksm1/Mxor_d<32>_xo<1>1 (ksm3/m2<32>)
     LUT6:I1->O            1   0.086   0.000  ksm3/Mxor_d<62>_xo<1>11 (ksm3/Mxor_d<62>_xo<1>1)
     MUXF7:I1->O           1   0.214   0.901  ksm3/Mxor_d<62>_xo<1>1_f7 (m3<62>)
     LUT6:I0->O            1   0.086   0.286  Mxor_d<121>_xo<1>1 (d_121_OBUF)
     OBUF:I->O                 2.144          d_121_OBUF (d<121>)
    ----------------------------------------
    Total                     13.387ns (3.998ns logic, 9.389ns route)
                                       (29.9% logic, 70.1% route)


=========================================================================
```

Fig8.7 Detailed analysis of proposed system

The simulation results of Montgomery ladder is shown in the figure 8.8 and karatsuba multiplier is shown in the figure 8.9.

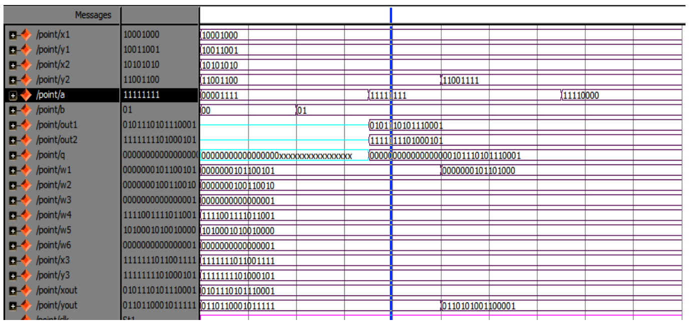

Messages
/point/x1 10001000
/point/y1 10011001
/point/x2 10101010
/point/y2 11001100
/point/a 11111111
/point/b 01
/point/out1 0101110101110001
/point/out2 1111111101000101
/point/q 000000000000000
/point/w1 0000000101100101
/point/w2 0000000100110010
/point/w3 0000000000000001
/point/w4 1111001111011001
/point/w5 1010001010010000
/point/w6 0000000000000001
/point/x3 1111111011001111
/point/y3 1111111101000101
/point/xout 0101110101110001
/point/yout 0110110001011111
/point/clk St1

Fig 3.8 Simulation results of Montgomery Ladder

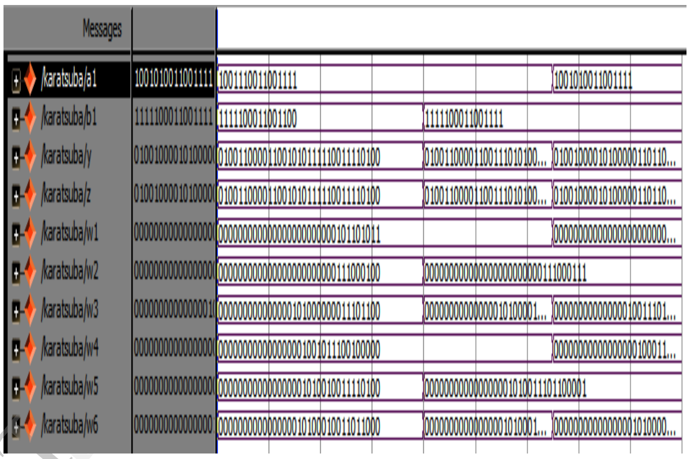

Messages
/karatsuba/a1 1001010011001111
/karatsuba/b1 1111100011001111
/karatsuba/y 010010000101000
/karatsuba/z 010010000101000
/karatsuba/w1 000000000000000
/karatsuba/w2 000000000000000
/karatsuba/w3 000000000000001
/karatsuba/w4 000000000000000
/karatsuba/w5 000000000000000
/karatsuba/w6 000000000000000

Fig 8.9 Simulation Results of Karatsuba Multiplier

Table 8.1 shows the comparison of existing and proposed system.

TABLE 8.1 COMPARISON TABLE OF EXISTING AND MODIFIED MULTIPLIER

LOGIC UTILIZATION	NUMBER OF SLICE LUTs		NUMBER OF FULLY UTILISED LUT-FF PAIRS		NUMBER OF BONDED IOBs	
	USED	AVAILABLE	USED	AVAILABLE	USED	AVAILABLE
EXISTING SYSTEM	15336	20480	0	15336	699	360
MODIFIED SYSTEM	4682	20480	0	4682	467	360

IX CONCLUSION

This proposed elliptic curve scalar multiplier used for high-speed and area-constrained applications. The implementation uses a novel pipelined bit-parallel Karatsuba multiplier that has sub quadratic complexity. A theoretical model was used to analyze delay of critical paths in the ECSMA and to obtain optimal pipelining. Efficient choice of scalar multiplication algorithm, optimized field primitives, balanced pipeline stages, and enhanced scheduling of point arithmetic resulted in a high-speed architecture with a significantly small area.

In future the power analysis can be done by using tools like TSpice and it can be reduced.

X. REFERENCES

[1] Wollinger.T, Guajardo.J, and Paar.C, “Security on FPGAs: State-of the-art implementations and attacks,” ACM Trans. Embedded Comput. Syst., vol. 3, no. 3, pp. 534–574, 2004.

[2] Roy.S.S , Rebeiro.C., and Mukho padhyay.D, “Theoretical modeling of the Itoh- Tsujii inversion algorithm for enhanced performance on k-LUT based FPGAs,” in Proc. Design, Autom., Test Eur. Conf. Exhibit., Mar.2011, pp. 1–6.

[3] López.J and Dahab.R., “Improved algorithms for elliptic curve arithmetician GF(2n),” in Proc. Sel. Areas Cryptography, 1999, pp. 201–212.

[4] Hankerson.D, Menezes.A.J, and Vanstone.S., Guide to Elliptic Curve Cryptography. Secaucus, NJ: Springer-Verlag, 2003.

[5] Joye.M. and Yen.S.M., “Optimal left-to-right binary signed-digit recoding,” IEEE Trans. Comput., vol. 49, no. 7, pp. 740–748, Jul. 2000.

[6] López.J. and Dahab.R., “Fast multiplication on elliptic curves over GF (2m) without pre computation,” in Proc. 1st Int. Workshop Cryptographic Hardw. Embedded Syst., 1999, pp. 316–327.

[7] Orlando.G. and Paar.C., “A high performance reconfigurable elliptic curve processor for GF (2m),” in Proc. 2nd Int. Workshop Cryptographic Hardw. Embedded Syst., 2000, pp. 41–56.

[8] Song.L. and Parhi.K.K, “Low-energy digit-serial/parallel finite field multipliers,” J. VLSI Signal Process., vol. 19, no. 2, pp. 149–166, Jul.1998.

[9] Gura.N., Shantz.S.C., Eberle.H,, Gupta.S., Gupta.V., Finchel stein.D.,Goupy.E., and Stebila.D., “An end-to-end systems approach to elliptic curve cryptography,” in Proc. 4th Int. Workshop Cryptographic Hardw. Embedded Syst., 2003, pp. 349–365.

[10] Järvinen.K., “Optimized FPGA-based elliptic curve cryptography processor for high speed applications,” Integr., VLSI J., vol. 44, no. 4,pp. 270–279, Sep. 2011.

[11] Solin.J.A as, “Efficient arithmetic on Koblitz curves,” Designs, Codes Cryptography, vol. 19, nos. 2–3, pp. 195–249, 2000.

[12] Järvinen.K. and Skytta.J., “On parallelization of high-speed processors for elliptic curve cryptography,” IEEE Trans. Very Large Scale Integr.(VLSI) Syst., vol. 16, no. 9, pp. 1162–1175, Sep. 2008.

[13] Azarderakhsh.R. and Reyhani-Masoleh.J., “Efficient FPGA implementations of point multiplication on binary Edwards and generalized Hessian curves using Gaussian normal basis,” IEEE Trans. Very LargeScale Integr. (VLSI) Syst., vol. PP, no. 99, p. 1, Jun. 2011.

[14] Saqib.N.A., Rodríguez-Henríquez.F., and Diaz-Perez.A., “A parallel architecture for fast computation of elliptic curve scalar multiplication over GF(2m),” in Proc. 18th Int. Parallel Distrib. Process. Symp., Apr.2004, pp. 144–151.

[15] Pu.Q. and Huang.J., “A micro coded elliptic curve processor for GF(2m)using FPGA technology,” in Proc. Int. Conf. Commun., Circuits Syst.,vol. 4. Jun. 2006, pp. 2771–2775.