International Journal of communication and computer Technologies, ISSN: 2278-9723

Available at http://www.ijccts.org

Analysis on FPGA Designs of Parallel High Performance Multipliers

Retheesh.D

Saveetha Engineering College

Received: 13-01-2014, **Revised:** 24-03-2014, **Accepted:** 29-04-2014, **Published online:** 21-06-2014

Abstract: For many applications from the areas of cryptography and coding, finite field multiplication is the most resource and time consuming operation. In this paper, optimized high performance parallel GF(2^{233}) multipliers for an FPGA realization were designed and the time and area complexities were analyzed. One of the multipliers uses a new hybrid structure to implement the Karatsuba algorithm. For increasing performance, we make excessive use of pipelining and efficient control techniques and use a modern state-of-the-art FPGA technology. As a result we have, to our knowledge, the first hardware realization of sub quadratic arithmetic and currently the fastest and most efficient implementation of 233 bit finite field multipliers.

1. INTRODUCTION

The arithmetic operations in finite fields are mainly used in cryptography and error control coding. Addition and multiplication are the two basic operations in the finite field GF(2^m).Addition in GF(2^m) is easily realized using m two-input XOR gates while multiplication is costly in terms of gate count and time delay. The other operations of the finite fields, such as exponentiation, division and inversion can be performed by repeated multi-plications.As a result there is a need to have a fast multiplication architecture with low complexities. The hardware/software implementation efficiency of finite field arithmetic is measured in terms of the associated space and time complexities. The space complexity is defined as the number of XOR and AND gates needed for the implementation of the circuit,whereas the time complexity is the total gate delay of the circuit. The space and time complexities of a multiplier heavily depend on how the field elements are represented. An element of GF(2^m) is usually represented with respect to one of the three popular bases: Polynomial (canonical or standard) basis(PB), dual basis (DB), and normal basis (NB).

Especially for the area of cryptography where the extension of the finite field *GF(2^m)* is fairly large, say $m > 160$, the selection of the multiplication algorithm has a major impact on the overall system performance. The selection of the finite field is based on the *FlPS 186-2* standard concerning with the digital signature algorithms and proposed by NIST. This standard suggests *5* binary fields, mainly the extension degrees *163*,233, 283**,** 409, and 573, which are all prime extensions. We have selected *GF(2^{233})* to satisfy the security requirements in elliptic curve cryptography for the next years, but our results can be adapted to finite fields with other prime extensions as well.

For cryptography, the requirements with respect to performance and security may change depending on the application. For this reason we use FPGAs as target technology in order to avoid the flexibility lacking in ASIC designs. It turns out that many opti-mizations of field multipliers proposed for ASIC design do not hold for FPGA. The main differences are

- Influence of routing on the FPGA performance.
- 4-input lookup table technology instead of 2-input logic gates.

- Treatment of high-fanout nets on FPGAs.

So we decided to create completely new FPGA optimized designs for the multipliers.

The paper is organized as follows. In the next section we give an overview over related work. Section 3 gives a short introduction into the theory of operation of the classical, Karatsuba and Massey-Omura multipliers. The architecture and FPGA implementation of these multipliers is described in detail in Section 4. The performance results and a comparison is given in Section *5*.

2. RELATED WORK

Several works concern the comparison of different hardware based multiplier architectures in the binary finite fields. The authors of [3] have compared three known serial multipliers, namely Berlekamp, Massey-Omura. and a polynomial basis multiplier. And implemented them for a small finite field $GF(2^8)$ in VLSI..[4] considers VLSI implementation of parallel multipliers for a class of finite fields $GF(2^m)$ with extension degrees m = 8, 16,24, and 32**.**which are not prime extension degrees and are believed to have security weaknesses**.** [6] considers different parallel multipliers in $GF(2^4)$ which is suitable for coding applications. This work also considers hardware optimization techniques to improve the performance of multipliers and make some estimates which hold only for small finite fields.[7] gives a detailed comparison of different VLSI implementations of parallel multipliers in $G\ F\ (2^4)$.Indeed all of the above works (except [4]) correspond to small finite fields and the results can not be easily extended to larger fields**.**

With the development of new FPGA families with large gate counts, however, it is possible to realize parallel finite field multipliers on a single chip which performs the total multiplication operation in a few clock cycles. So it become necessary to have a performance analysis of the multipliers for large field extensions (with m>160)to select the best multiplier for a certain application.

3. MULTIPLICATION IN THE BINARY FINITE FIELDS

There are several algorithms to multiply two finite field elements and each of them has its benefits depending on the finite field size, the implementation type (hardware or software), and the time and area requirements. One of the main differences between these algorithms is the finite field representation basis. In this section we give a brief introduction of different hardware based finite field multipliers in $GF(2^m)$ along with their space and time complexities. When the Hamming weight of the irreducible polynomial plays a significant role, we assume the existence of an irreducible trinomial of degree n when considering the multiplication in $GF(2^m)$.This is a reasonable assumption since our special finite field is $GF(2^{233})$**,** and the polynomial $x^{233} + x^{74} + 1$ is irreducible. On the other hand it is conjectured that a trinomial of degree m exists for a large amount of values n. Multipliers will be categorized depending on the finite field basis.

3.1. Normal Basis Multipliers

An element a in $GF(2^m)$ is called a normal element, when the elements of the set

$\Gamma = \{a^{2^\wedge i} | 0 <= i < m)$ are linearly independent. In this case, the set Γis called a normal basis. One great advantage of the normal bases is that squaring in this basis consists of only a cyclic shift (which requires no logic elements and can he done in nearly zero time). There are two types of normal bases for which there exist effective multiplication methods, namely optimal normal bases of type I and II.

It is well-known that there always exists a normal basis in the field *GF* (2^m) over *GF(2)* for all positive integers m .By finding an element β ∈ *GF*(2^m)such that

$$\{\beta,\beta^2,\text{------},\beta^{2m-1}\}$$

is a basis of GF(2^m) over GF(2), any element A Є GF(2^m) can be represented as

$$A = \sum_{i=0}^{m-1} \alpha_i\beta^{2^\wedge i} = \alpha_0\beta+\alpha_1\beta^2+\text{----}+\alpha_{m-1}\beta^{2m-1} \quad , (1)$$

Where α^i Є *GF(2)*, $0 \leq i \leq m-1$, is the *i*th coordinate of A with respect to the NB. In short, the normal basis representation of A will be written as

$$A= (\alpha_0, \alpha_1, \text{----}, \alpha_{m-1}).$$

In vector notation, however, (1) can be written as

$$A = \underline{\alpha} \times \underline{\beta}^T = \underline{\beta} \times \underline{\alpha}^T ,$$

Where $\underline{\alpha} =[\alpha_0, \alpha_1,\ldots,\alpha_{m-1}]$, $\underline{\beta}= [\beta,\beta^2, \ldots\beta^{2m-1}]$, and T denotes vector transposition.

The main advantage of the NB representation is that an element A can be easily squared by applying right cyclic shift of its coordinates, since

$$A^2= (\alpha_{m-1}, \alpha_0,\text{-----} ,\alpha_{m-2}) = \alpha_{m-1}\beta +\alpha_0\beta^2+\text{------}\alpha_{m-2}\beta^{2m-1}.$$

3.1.1.Massey-Omura Parallel Multiplier

The Massey-Omura multiplier is one of the most famous multipliers that work in the normal basis representation. It consists of similar blocks which can work in parallel to generate output bits simultaneously. One great advantage of this multiplier is its flexibility as a serial -parallel multiplier. This means that the designer has the ability to select an arbitrary number of similar blocks to achieve different numbers of output bits in one clock cycle, depending on the given area constraints. For the case of optimal normal bases in *GF*(2^m) one requires $(2m-2)D$ 2-input XOR and m D 2-input XOR gales, where D is the number of output bits per clock cycle. The minimum combinatorial propagation delay is $T_{AND} + [\log_2 n]T_{XOR.}$

Let A and B be two elements of *GF*(2^m) and represented with respect to the NB as

$$A = \sum_{i=0}^{m-1} \alpha_i\beta^{2^\wedge i} \quad \text{and}$$

$$B = \sum_{j=0}^{m-1} b_j \beta^{2^\wedge i} ,$$

respectively. Let C denote their product as

$$C=AB= (\underline{\alpha}\times\underline{\beta}^T)\times(\underline{\beta}\times\underline{b}^T)=\underline{\alpha} \times\underline{M}\times\underline{b}^T, \quad (4)$$

Where the multiplication matrix M is defined by

$$M = \underline{\beta}^T \times \underline{\beta} = [\beta^{2^\wedge i+2^\wedge j}] =$$

$$\begin{bmatrix} \beta 20+20 & \beta 20+21 & \ldots\ldots & \beta 20+2m-1 \\ \beta 21+20 & \beta\ 21+21 & \ldots\ldots. & \beta 21+2m-1 \\ . & . & & . \\ . & . & & . \\ . & . & & . \\ \beta 2m-1+20 & \beta 2m-1+21 & \ldots\ldots & 2m-1+2m-1\beta \end{bmatrix} \quad (5)$$

If all entries of M are written with respect to the NB, then the following is obtained

$$M=M_0\beta+M_1\beta^2+\text{-----}+M_{m-1}\beta^{2m-1}, \quad (6)$$

where M_is are m x m matrices whose entries belong to GF(2). By substituting (6) into (4), the coordinates of C are found as follows:

$$c_i = \underline{\alpha} \times M_i \times \underline{b}^T, \qquad 0 \le i \le m-1,$$
$$= \underline{\alpha}^{(i)} \times M_0 \times \underline{b}^{(i)T}, \qquad 0 \le I \le m-1, \quad (7)$$

where $\underline{\alpha}^{(i)} = [\alpha_i, \alpha_{i+1}, \ldots\ldots \alpha_{i-1}]$
and $b(i) = [b_i, b_{i+1}, \ldots.., b_{i-1}]$

are,respectively, the i-fold left cyclic shift of a and b . It is not difficult to verify that the number of 1s in each Mi, 0 <= i <= m – 1, is the same, which is here after denoted as C_N. Since these nonzero entries of Mi determine the gate count of the normal basis multiplier, C_N is referred to as the complexity of the NB .

The coordinate c_i in (7) can be written as modulo 2 sum of exactly C_N terms. Each of these terms is a modulo 2 product of exactly two coordinates (one of A and B each). Thus, the generation of c_i requires C_N multiplications and C_{N-1} additions over GF(2). In hardware, this corresponds to C_N AND gates and (C_{N-1}) XOR gates, assuming that all gates have two inputs. If these XOR gates are arranged in the binary tree form, then the total gate delay to generate c_is $T_A + [\log_2 C_N\ d]T_X$, where T_A and T_X are the delays of one AND gate and one XOR gate, respectively. For parallel generation of all c_is, i = 0, 1,------, m- 1, one needs mC_N AND and $m(C_N – 1)$ XOR gates. Also, one can reduce the number of AND gates to m^2 by reusing multiplication terms over GF(2). Thus, to reduce the number of XOR gates, we have to choose a normal basis such that CN is minimum. It was proven that $C_N \ge 2m - 1$.If $C_N = 2m - 1$, then the NB is called an optimal normal basis (type-I or type-II).

3.2. Polynomial Basis Multipliers

In this basis, each element is represented as a linear combination of different powers of a root of an irreducible polynomial. Indeed multiplication in this basis consists of a polynomial multiplication followed by a modular reduction. There are different possibili- ties to multiply two elements in this basis like the Mastrovito, the classical, and the Karatsuba multipliers. Since there is only small difference in time and space complexities of the Mastrovito and the classical multipliers we select the classical multiplier because of its regular structure and the possibility of pipelining which is difficult to apply to the Mastrovito multiplier..

3.2.1 Classical Multiplier

The most straight forward method to perform finite field multiplication is to multiply the polynomials and then reduce the result modulo an irreducible polynomial to achieve the final result. The school method polynomial multiplication requires n2 *AND* gates and $(n - 1)^2$ XOR gates (2-input each). The combinatorial propagation delay across a school method multiplier is $T = T_{AND} + [\log_2\ n]T_{XOR}$. Reducing modulo the polynomial *f (x)* can be done using *(r* - l)(n - 1) two input XOR gates, where *T* and n are the Hamming weight and the degree of the polynomial f (x) , respectively.

3.2.2.Karatsuba Multiplier

An approach to reduce the number of gates in the polynomial basis multipliers is the Karatsuba method**.** In this method the number of multiplications is reduced but at the cost of increasing the number of additions and the total propagation delay. This method decreases the total number of gates from O (n^2) to the O $(n^{1.59})$, which is very effective when the polynomials become large**.** To achieve a tradeoff between the area and propagation delay which is long in the Karatsuba multipliers, we have used a hybrid structure by using the Karatsuba multiplication formulas(see [l0]) for the polynomials of degree 1 and 2 in a hierarchical manner above school method multipliers of degree 39**.** This structure requires 28800 AND and 31183 XOR gates, and a total propagation delay of T_{AND} + l4T_{XOR}.. The costs for a pure Karatsuba multiplier are 6561 AND, 37320 XOR, and T_{AND} + 26T_{XOR} and for a school method multiplier are 54289 AND, 53824 XOR, and $T_{AND} + 8\,T_{XOR}$.

Let the field GF(2^m) be constructed using the irreducible polynomial P(x) of degree $m=rn$, with r=2^k,k an integer. Let A,B be two elements in GF(2^m). Both elements can be represented in the polynomial basis as,

$$A = \sum_{i=0}^{m-1} \alpha_i x^i = \sum_{i=m/2}^{m-1} \alpha_i x^i + \sum_{i=0}^{m/2-1} \alpha_i x^i$$

$$= x^{m/2} \sum_{i=0}^{m/2-1} \alpha_i{+}m/2x^i + \sum_{i=0}^{m/2-1} \alpha_i x^i = x^{m/2} A^H + A^L$$

and

$$B = \sum_{i=0}^{m-1} b_i x^i = \sum_{i=m/2}^{m-1} b_i x^i + \sum_{i=0}^{m/2-1} b_i x^i$$

$$= x^{m/2} \sum_{i=0}^{m/2-1} b_i{+}m/2x^i + \sum_{i=0}^{m/2-1} b_i x^i = x^{m/2} B^H + B^L$$

Then, using last two equations, the polynomial product is given as

$$C=x^m A^H B^H+(A^H B^L+A^L B^H)\, x^{(m/2)} + A^L B^L. \quad (3)$$

Karatsuba algorithm is based on the idea that the product of last equation can be equivalently written as

$$C = x^m A^H B^H + A^L B^L + (A^H B^H + A^L B^L + (A^H + A^L)(B^L+B^H))\, x^{m/2}$$

$$= x^m C^H + C^L. \quad (4)$$

Let us define

$$M_A := A_H+A_L;$$
$$M_B := B_L+B_H;$$
$$M := M_A M_B; \quad (5)$$

Using equation (4),and taking into account that the polynomial product C has at most *2m-1* coordinates, we can classify it coordinates as

$$C^H = [c_{2m-2}, c_{2m-3}, ----, c_{m+1}, c_m]$$
$$CL = [c_{m-1}, c_{m-2}, -----, c_1, c_0]. \quad (6)$$

Although (4) seems to be more complicated than (3), it is easy to see that equation (4) can be used to compute the product at a cost of four polynomial additions and three polynomial multiplications. In contrast, when using equation (3), one needs to compute four polynomial multiplications and three polynomial additions. Due to the fact that polynomial multiplications are in general much more expensive operations than polynomial additions, it is valid to conclude that (4) is computationally simpler than the classic algorithm. Karatsuba's algorithm can be applied recursively to the three polynomial multiplications in (4). Hence, we can postpone the computations of the polynomial products $A^H B^H$, $A^L B^L$ and M, and instead we can split again each one of these three factors into three polynomial products. By applying this strategy recursively, in each iteration each degree polynomial multiplication is transformed into three polynomial multiplications with their degrees reduced to about half of its previous value.

Input: Two element $A,B \in GF(2^m)$ with $m=rn=2^k n$,and where A,B can be expressed as,
$A=x^{m/2} A^H+A^L$, $B=x^{m/2} B^H+B^L$
Output: A polynomial $C=AB$ with up to *2m-1* coordinates,
where $C=x^m C^H+C^L$.
Procedure *K*mul2^k *(C, A, B)*

```
0. begin
1. if $(r==1)$ then
2. $C=mul\_n(A,B);$
3. return;
4. for $i$ from $0$ to $r/2-1$ do
5. $M_{Ai}=A_i^L+A_i^H;$
6. $M_{Bi}=B_i^L+B_i^H;$
7. end
8. $mul2^k(C^L,A^L,B^L);$
9. $mul2^k(M,M^A,M^B);$
10. $mul2^k(C^H,A^H,B^H);$
11. for $i$ from $0$ to $r-1$ do
12. $M_i = M_i+C_i^L+C_i^H;$
13. end
14. for i from 0 to r-1 do
15. $C_{r/2+I} = C_{r/2+i}+M_i;$
16. end
17. end
```

Fig (a).m=2^kn-bit Karatsuba multiplier.

The algorithm presented in figure 1 implements theKaratsuba strategy for polynomial multiplication. It can be shown that the space and time complexities of that algorithm are given as,

$$\#XORs \leq (m/n)^{\log_2 3}(8m/r-2+M_{xor2}{}^n) -8m+2; \quad (7)$$
$$\#AND \leq (m/n)^{\log_2 3}M_{and2}{}^n;$$
$$Delay \leq T_{delay2}{}^n + 4Tx\log_2(m/n).$$

In this case it has been assumed that the block selected to implement the $GF(2^n)$ arithmetic has a $T_{delay\ 2}{}^n$ gate delay associated with it.

As it has been mentioned above, the hybrid approach proposed here requires the use of an efficient multiplier algorithm to perform the n-bit polynomial multiplications. It can be shown that the space and time complexities for the classic ö n-bit multiplier are given as

$$\#XORs = (n-1)^2;$$
$$\#ANDs = n^2;$$
$$Delay \leq T_{AND} +T_X[Log_2 n]. \quad (8)$$

Combining the complexities given in equation (8), together with the complexities of equation (7) we conclude that the space and time complexities of the hybrid *m* bit Karatsuba multiplier truncated at the n-bbit multiplicand level are upper bounded by

$$\#XORs \leq (m/n)^{\log_2 3}(n^2+6n-1)-8m+2;$$
$$\#AND \leq 3^{\log_2 r}M_{and2}{}^n =(m/n)^{\log_2 3}n^2; \quad (9)$$
$$Delay \leq T_{AND}+T_X(\log_2 n+4\log_2 r).$$

4. FPGA IMPLEMENTATIONS OF PARALLEL MULTIPLIERS

In this section we have presented the architectures of the parallel multipliers. The interface logic is the same for all multipliers so we can use the same test bench and the designs are interchangeable.

4.1. **Massey-Omura Multiplier**

If implemented fully in parallel, the resource requirements of the Massey-Omura multiplier are very large (exceeding the LUP resources of our FPGA by about 7 percent), but it can be realized with any degree of parallelism between fully parallel and fully serial. So we use a semi-parallel implementation where a multiplication is performed in two steps.

As shown in Figure 1, Massey-Omura consists of two cycshift stages' with 117 outputs each. Output n is the same as output *n* - 1 but cyclically rotated by one bit. The 117 rotated operand pairs are passed in parallel to 117 identical XOR trees (XOR.1 ... XOR-117) that compute the lower 117 hits of the result. The last outputs of the cycshift stages are fed back to the inputs via an operand register, so the second set of rotated operands as well as the higher pan of the result is generated one clock cycle later.

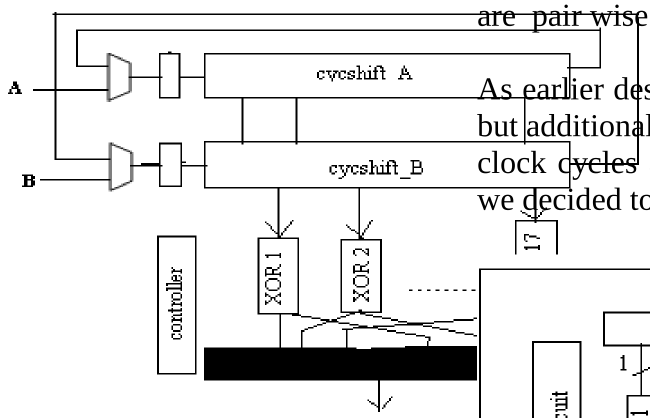

Fig.1.Semi-parallel Massey-Omura multiplie

4.2.The Classical Multiplier

The implemented classical multiplier consists of a polynomial multiplier followed by the modular reducer as shown in Figure 2. Assuming that the polynomial

$C=C_{2n-2}x^{2n-2}+c_{2n-3}x^{2n-3}+\text{----}+c_1x+c_0$

Is the product of two polynomials $a=a_{n-1}x^{n-1}+a_{n-2}x^{n-2}+\text{----}+a_1x+a_0$ *and* $b=b_{n-1}x^{n-1}+b_{n-2}x^{n-2}+\text{----}+b_1x+b_0$, then different coefficients of c can be computed using the equation (1)

$$
\begin{aligned}
c_0 &= a_0b_0\\
c_1 &= a_0b_1+a_1b_0\\
&\text{-------}\\
c_{n-1} &= a_0b_{n-1}+a_1b_{n-2}+\text{-----}+a_{n-2}b1+a_{n-1}b_0 \qquad (1)\\
&\text{--------}\\
c_{n-3} &= a_{n-2}b_{n-1}+a_{n-1}b_{n-2}\\
c_{n-2} &= a_{n-1}b_{n-1}
\end{aligned}
$$

Each of the rows of **(1)** has some elements which must be combined in a XOR tree to generate a single bit of the result. The rows c_i and c_{2n-2-i} for $0<=i<n-1$ are generated with tree structured XOR-circuits of identical length, but with different in- puts. So we have a total of 465 XOR trees, where 464 of them are pair wise equal in size.

As earlier design of ours which used only 233 trees but additionally 232 multiplexers required more clock cycles and exceeded the FPGA resources, so we decided to use the full number of XOR trees.

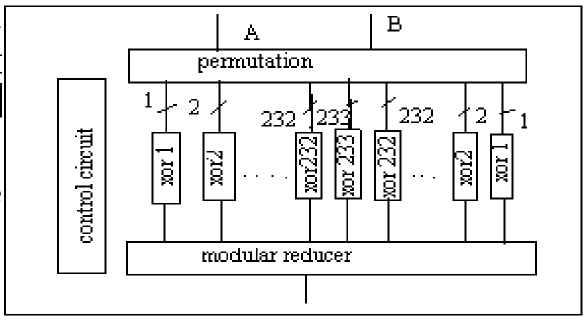

Fig.2. Classical multiplier for 233 Bit

4.3.The Hybrid Karatsuba Multiplier

We have used a hybrid structure to combine the Karatsuba algorithm with 2 and 3 coefficients respectively to generate a Karatsuba algorithm with 6 coefficients. Futhermore, we have used a new distributed control structure to implement the polynomial multiplication. The combination of these two Karatsuba methods has already been proposed in for composite extension finite fields.and for the Optimal Extension fields. But to our knowledge, it is the first time that such a combination has been implemented in hardware for prime extension finite fields. The block diagram of the complete multiplier is shown in Figure 3

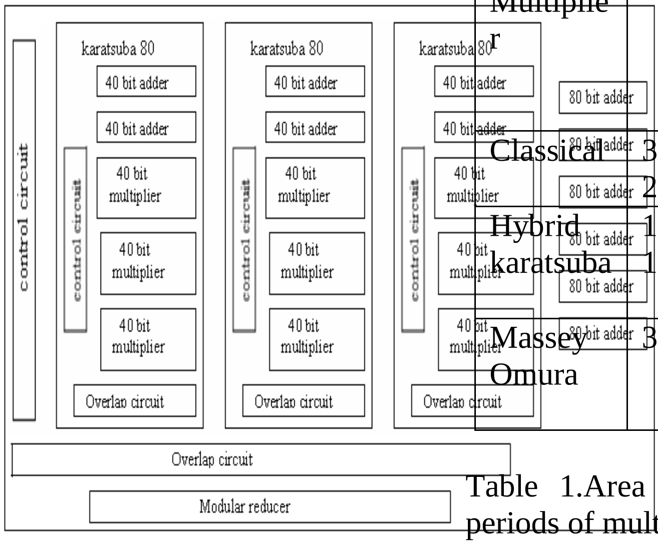

Fig.3.Hybrid parallel Karatsuba multiplier for 233 bit

The multiplier in the upper level consists of three 80-bit adders, and overlap circuit. Each of the multipliers will be used twice during a polynomial multiplication to cover the total six 80-bit 80-bit multiplications. The control circuit starts the multipliers at the at the suitable time tomake use of the pipelinestages in the multipliers. It also controls the timing of the adders. Since outputs of the different multipliers have some powers of z in common, the overlap circuit XORs the overlapping powers.

5.CONCLUSION

In this section the. performance comparison of the FPGA synthesis results were given. All multipliers are synthesized for a Xilinx xc2v-6000-ff1517-4 FPGA without pin mapping and area constraints. In subsequent synthesis iterations, we specified timing constraints with slightly increasing stringency in order to converge to an optimal timing. It should be noted that the clock cycle time is computed including the pad delays since all multipliers are implemented as "stand alone" designs.

Multiplier	LUT/FF	Equivalent Gate count	Clock period(freq)
Classical	37296/37552	528427	~13.00ns (~77Mhz)
Hybrid karatsuba	11746/13941	289489	11.07ns (90.33Mhz)
Massey Omura	36857/8543	608149	15.91ns (62.85Mhz)

Table 1.Area requirements and minimum clock periods of multipliers.

Table I gives a comparison of the number of 4-input LUTs, the number of flipflops, the equivalent gate count and the clock period for each multiplier.

7.REFERENCES

[1] Arash and Anwar “A new construction of Massey Omura parallel multiplier over GF(2^m)”, “ IEEE Transactions on computers, Vol 51 No.5,May 2002”.

[2] Henriquez & Koc “On fully parallel karatsubMultipliers for GF(2^m)”, in proceeding(394) Computer sciences and technology.Cancum,Mexico:ACTA Press,2003

[3] I. *S* Hsu. T. K. Truong. L. J. Deutsch. And I. *S* . Reed, “Acomparition of VLSI architecture of finite field multipliers using dual, normal. Or standard basis." IEEE Trunsucrionr *on Computer.*<. vol. 37. no. 6. DO. 735-739.

[4] Niegel. P. Sman. "How Secure Are Elliptic CUNW over CompositeExtension Fields?," in *Advuncss* in *Crypmlugy:* Proceedings 6EU-*ROCRYPT2001*, Aarhus. Denmark, B. Pfilzmann. Ed. 2001, number2045 in Lecture Notes in Computer Science. pp. 3&39. Springer-Verlvg.

[5] C. Gregory. C. Ahlquist. B. Nelson, and M. Rice. "Optimal finitefields for FPGAs:' in Proceedings *offhe* 91h *Inrrmuiionul* Worbhop*on Field Pmgrummoble h g i c and Applications (FPL 99J.* Glasgow.UK. AUKUSI 1999, number 1613. DD. 51-61, Sorineer. . .. . –