International Journal on Recent and Innovation Trends in Computing and Communication ISSN: 2321-8169
Volume: 5 Issue: 12 145 – 149

Analysis of *GF* (2^m) Multiplication Algorithm: Classic Method v/s Karatsuba-Ofman Multiplication Method

Manish Kumar Goyal
Lecturer, Government Polytechnic College, Bagidora
manishbmgoyal@gmail.com

Shiv Karan Meghwal
Lecturer, Government Polytechnic College, Dungarpur
shivkaran_sonel@yahoo.co.in

Abstract **-** In recent years, finite field multiplication in GF(2^m) has been widely used in various applications such as error correcting codes and cryptography. One of the motivations for fast and area efficient hardware solution for implementing the arithmetic operation of binary multiplication , in finite field GF (2^m), comes from the fact, that they are the most time-consuming and frequently called operations in cryptography and other applications. So, the optimization of their hardware design is critical for overall performance of a system. Since a finite field multiplier is a crucial unit for overall performance of cryptographic systems, novel multiplier architectures, whose performances can be chosen freely, is necessary. In this paper, two Galois field multiplication algorithms (used in cryptography applications) are considered to analyze their performance with respect to parameters viz. area, power, delay, and the consequent Area×Time (AT) and Power×Delay characteristics. The objective of the analysis is to find out the most efficient GF(2^m) multiplier algorithm among those considered.

*****

1. INTRODUCTION

A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing a set of partial products, and then summing the partial products together. This method, involving partial products is mathematically correct, but it has the following two serious engineering problems:

1. It involves 32 intermediate additions in a 32-bit computer, or 64 intermediate additions in a 64-bit computer. These additions take a lot of time. The engineering implementation of binary multiplication consists, in fact, of simplification of the mathematical process and compromising with the increased complexity hence enforced, in order to do fewer additions. When implemented in software, long multiplication algorithms have to deal with overflow during additions, which can be expensive.

2. The basic school method handles the sign with a separate rule ("+ with + yields +", "+ with - yields -", etc.). Modern computers embed the sign of the number in the number itself, usually in the two's complement representation. That forces the multiplication process to be adapted to handle two's complement numbers, complicating the process a bit more.

To multiply two numbers with *n*-digits using this method, one needs about n^2 operations. More formally: using a natural size metric of number of digits, the time complexity of multiplying two *n*-digit numbers using long multiplication is $\Theta(n^2)$.Therefore, to solve above problem modular multiplication can be used. The modular exponentiation applies modular multiplication repeatedly. Modular multiplication is a mathematical operation on integer *A.B mod M* with *A, B<M* where by *A* and *B* are the operand and *M* is moduls. Modular multiplication *A×B mod M* can be performed in two different ways:

- Multiplying, i.e. computing *P = A×B*; then reducing, i.e. *R = P mod M*
- Interleave the multiplication and the reduction steps.

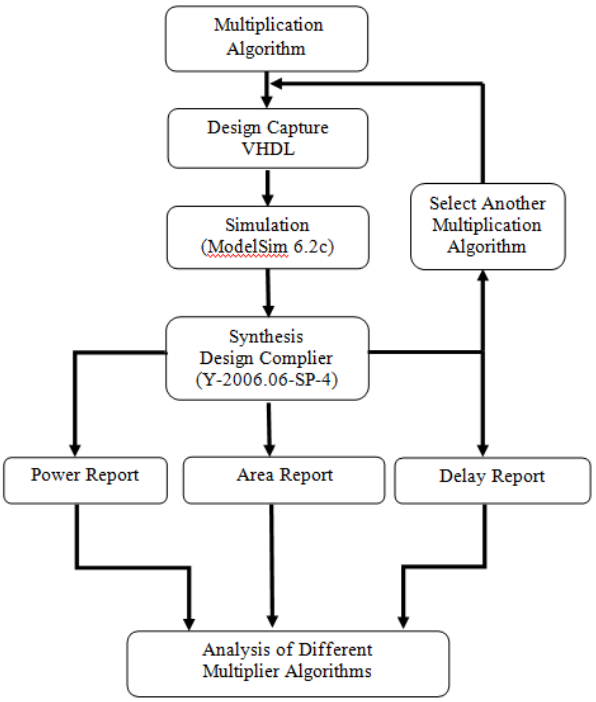


Figure-1.1: Flow-chart of steps undertaken for analysis of different Multiplier Algorithms

So the performance of systems based on modular multiplication, e.g. the public key Cryptosystem, is primarily

determined by the implementation efficiency of the modular multiplication and exponentiation. As the operands are usually large (i.e. 1024 bits or more), it is necessary to improve upon the computation time of the encryption/decryption operations. Hence, it is essential to minimize the number of modular multiplications performed and to reduce the time required by a single modular multiplication . There are various algorithms that implement modular multiplication. Here following Two algorithms have been considered:

1 Classic Multiplier
2 Karatsuba-Ofman multiplier

Starting with each of the above mentioned algorithms, the following methodology has been adopted for the analysis of a particular design based on design parameters viz. area-on-chip, power consumption and related dlays. Subsequent comparison among the various designs based on the extracted values of the design parameters mentioned earlier has been provided

2. THE BASIC MODULAR MULTIPLIER: TWO-STEP CLASSIC MULTIPLIER

The two-step classic multiplication in $GF(2^m)$ is a straightforward translation of the classic school multiplication algorithm. This method has an asymptotic complexity $O(n^2)$ [6]. In the two-step multiplication, Let $a(x)$ and $b(x)$ be two field elements, the field product $c(x)$ given in Eq. (2.1).

$$c(x) = a(x)b(x) \bmod f(x) \quad \ldots\ldots (2.1)$$

Where,

$$f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0 \quad \ldots\ldots (2.2)$$

And, $f_i \in GF(2) = \{0, 1\}$, the set $\{1, x, \ldots, x_{m-1}\}$ is polynomial basis in $GF(2^m)$.

2.1 FUNDAMENTAL CONCEPT

This involves two steps for multiplication

1. Polynomial multiplication
2. Reduction modulo an irreducible polynomial [5]

2.1.1 POLYNOMIAL MULTIPLICATION

The product $d(x)$ of the polynomials $a(x)$ and $b(x)$, i.e. $d(x) = a(x)b(x)$, is a polynomial with maximum degree $(2m - 2)$. Polynomial multiplication $d(x)$ can be written in matrix form [3]. The coefficients of $d(x)$ are determined by the following expression:

$$d_k = \begin{cases} \sum_{i=0}^{k} a_i b_{k-i}\,; k = 0, \ldots, m-1 \\ \sum_{i=k}^{2m-2} a_{k-i+(m-1)} b_{i-(m-1)}\,; k = m, \ldots, 2m-2 \end{cases} \quad \ldots (2.3)$$

This expression have addition and multiplication in $GF(2)$. Assume that the following two functions can compute addition and multiplication for *mod 2*, using logical operation *x XOR y* and *x AND y* respectively.

function m2xor(x, y: bit) return bit

function m2and(x, y: bit) return bit

The total gate complexity for the bit-parallel computation of the matrix-vector product given in Eq. (2.3) is m^2 *AND* gates and $(m-1)^2$ *XOR* gates. The *AND* gates operate all in parallel and require a single *AND* gate delay T_{AND}, while the *XOR* gates are organized as a binary tree of depth $\lceil \log_2 j \rceil$ in order to add j operands. The total time complexity is then found by considering the largest number of terms, which is equal to m for the computation of d_{m-1}. Therefore, the total delay complexity for the bit parallel matrix-vector product is $T_{AND} + \lceil \log_2 j \rceil T_{XOR}$.

2.1.2 REDUCTION MODULO AN IRREDUCIBLE POLYNOMIAL

After the polynomial multiplication $d(x) = a(x).b(x)$, the next step is a reduction modulo an irreducible polynomial $f(x)$ must be performed. In modular reduction $c(x) = d(x) \bmod f(x)$, the degree $(2m - 2)$ polynomial $d(x)$ is reduced by the degree m irreducible polynomial $f(x)$, resulting in a polynomial $c(x)$ with degree $\deg(c(x)) \leq (m - 1)$.

$$\begin{aligned} c(x) &= d(x) \bmod f(x) \\ &= (d_{2m-2}x^{2m-2} + \ldots\ldots + d_1x + d_0) \bmod f(x) \\ &= c_{m-1}x^{m-1} + \ldots + c_1x + c_0 \end{aligned} \quad \ldots (2.4)$$

Reduction modulo $f(x)$ can be viewed as a linear mapping of the $(2m - 1)$ coefficients of $d(x)$ into the m coefficients of $c(x)$. Matrix consists of an $(m \times n)$ identity matrix and an $(m \times m - 1)$ matrix **R** named *reduction matrix*. The **R** matrix is a function only of the irreducible polynomial $f(x)$ as in Eq (2.5). Therefore, a reduction matrix **R** is uniquely assigned to every $f(x)$. The $r_{j,i} \in GF(2)$ coefficients of **R** can be recursively computed in function of $f(x)$ as follows:

$$r_{j,i} = \begin{cases} f_j\,; j = 0, \ldots, m-1; i = 0 \\ r_{j-1,i-1} + r_{m-1,i-1}r_{j,0}\,; j = 0, \ldots, m-1; i = 1, \ldots, m-2 \end{cases}$$

$\ldots.$ (2.5)

Where, $r_{j-1,i-1} = 0$; if j = 0.

R is function of the selected irreducible polynomial. Therefore, by choosing an appropriate reduction polynomial $f(x)$ the complexity of this operation can be reduced. The following function can compute the reduction matrix **R**

Function

reduction_matrix_R(f: poly_vector) return poly_matrix_m1m2

Where,

poly_matrix_m1m2 is an $(m \times m - 1)$ matrix of bits.

Finally, the two-step classic multiplication performing $c(x) = a(x)b(x) \bmod f(x) = d(x) \bmod f(x)$ using Eq.(2.4) and the reduction matrix computed with Eq.(2.5) can be given, where the previously defined functions *poly_multiplication* and *reduction_matrix_R* are used.

2.2 SIMULATION , SYNTHESIS AND RESULTS

A VHDL model for the classic multiplication algorithm has two components *poly_multiplier* and *poly_reducer* that implement the polynomial multiplication and the reduction modulo $f(x)$, respectively. The VHDL code for each value of m has been synthesized using Design Compiler with 180*nm* UMC library. The script for Design Compiler has been written so as to include the capability to generate reports for Power, Time and Area of the synthesized design. The procedure has been repeated for different values of m, and the values of each of the design parameters, hence obtained, has been provided in tabular format below.

Table-2.1: For different m Power requirement of Classic Multiplier Algorithm

m	Cell Internal Power (*mW*)	Net Switching Power (*mW*)	Total Dynamic Power (*mW*)	Cell Leakage Power (*μW*)	Total cell Area	Data Arrival Time (*ns*)
8	3.9246	1.8381	5.7627	0.0272528	4119.096191	0.81
16	11.3900	5.6327	17.0227	0.0746353	11695.97754	1.07
32	52.6769	28.8272	81.5041	0.3519799	54402.57813	1.45
64	234.1922	208.9930	443.1851	1.4260	223284.0156	1.95
128	909.1452	1690.3625	2599.5077	5.429	854699.1563	4.283
163	1371.254	3473.258	4844.512	8.295	1301869.125	7.943
233	2225.3654	102960.145	105185.5104	14.83	2295698.589	24.471

3. KARATSUBA-OFMAN MULTIPLICATION

The *Karatsuba-Ofman* algorithm is a recursive method for efficient polynomial multiplication or efficient multiplication in positional number systems. It has recursive application of the divide-and-conquer, thus, Karatsuba algorithm leads to a running time of $O(n^{\log_2 3}) \approx O(n^{1.585})$ for $n = 2^i$ $(i > 0)$ [6]. It is known that two arbitrary polynomials in one variable of degree less or equal to $(m - 1)$ with coefficients from a field $GF(2^m)$ can be multiplied with not more than m^2 multiplications in $GF(2^m)$ and $(m - 1)^2$ additions in $GF(2^m)$.

3.1. FUNDAMENTAL CONCEPT

The Karatsuba-Ofman algorithm provides a recursive algorithm which reduces the above multiplicative and additive (for large enough m) complexities [12]. A Karatsuba-Ofman algorithm restricted to polynomials, where $m = 2^t$ with t an integer, Let $a(x)$ and $b(x)$ be two elements in $GF(2^m)$. We are interested in finding the product $d(x) = a(x)b(x)$, with degree $\leq (2m - 2)$. Both elements can be represented in the polynomial basis as follow [10][14] :

$a(x)$

$= x^{m/2}(x^{m/2-1}a_{m-1} + \ldots + a_{m/2}) + (x_{m/2-1}\, a_{m/2-1} + \ldots + a_0)$

$= x^{m/2} A_H + A_L$ …… (3.1)

$b(x)$

$= x^{m/2}(x^{m/2-1}b_{m-1} + \ldots\ldots + b_{m/2}) + (x_{m/2-1}\, b_{m/2-1} + \ldots\ldots + b_0)$

$= x^{m/2} B_H + B_L$ ……. (3.2)

Using above expression, the polynomial product is given as
$d(x) = x^m A_H B_H + x^{m/2}(A_H B_L + A_L B_H) + A_L B_L$ ……. (3.3)
Let us define the following auxiliary polynomials

$M_0^{(1)} = A_L(x)\, B_L(x)$ …… (3.4)

$M_1^{(1)} = [A_L(x) + A_H(x)][B_L(x) + B_H(x)]$ …… (3.5)

$M_2^{(1)} = A_H(x)B_H(x)$ …… (3.6)

Then the product is given by:

$d(x)$
$= x^m M_2^{(1)}(x) + x^{m/2}[M_1^{(1)}(x) + M_0^{(1)}(x) + M_2^{(1)}(x)] + M_0^{(1)}(x)$
…… (3.7)

The algorithm becomes recursive if it is applied again to the polynomials given in Eq. (3.3) [10]. The next iteration step splits the polynomials A_L, B_L, A_H, B_H, $(A_L + A_H)$, and $(B_L + B_H)$ again in half [11]. With these newly halved polynomials, new auxiliary polynomials $M^{(2)}(x)$ can be defined in a similar way to Eq. (3.6). The algorithm eventually terminates after t steps. In the final step the polynomials $M^{(t)}(x)$ are degenerated into single coefficients. Since every step halves the number of coefficients, the algorithm terminates after $t = log_2 m$ steps [9][10].

3.2 SIMULATION, SYNTHESIS AND RESULTS

The VHDL code for the *Karatsuba-Ofman* algorithm has been simulated on the ModelSim-6.2c for different values of the *m*. The corresponding results obtained post-simulation were found to match the theoretical results. The VHDL code for each value of *m* has been synthesized using Design Compiler with 180*nm* UMC library, following results have been obtained.

Table–3.1: For different *m* Power requirement of Karatsuba-Ofman multiplier

m	**Cell Internal Power (*mW*)**	**Net Switching Power (*mW*)**	**Total Dynamic Power (*mW*)**	**Cell Leakage Power (*nW*)**	**Total cell Area**	**Data Arrival Time (*ns*)**
8	0.8923746	0.2291768	1.1216	4.4110	599.960938	0.32
16	1.0196	0.278.8957	1.2985	5.0031	677.375977	0.33
32	1.8970	0.4995943	2.3966	9.3389	1270.885132	0.32
64	3.9009	1.0411	4.9420	19.195	2612.731689	0.32
128	7.9043	2.1214	10.0257	38.9081	5296.433594	0.32
163	15.98	4.282	20.262	78.58	10629.8964	0.32
233	20.3058	5.4825	25.7883	99.8702	13592.66992	0.33

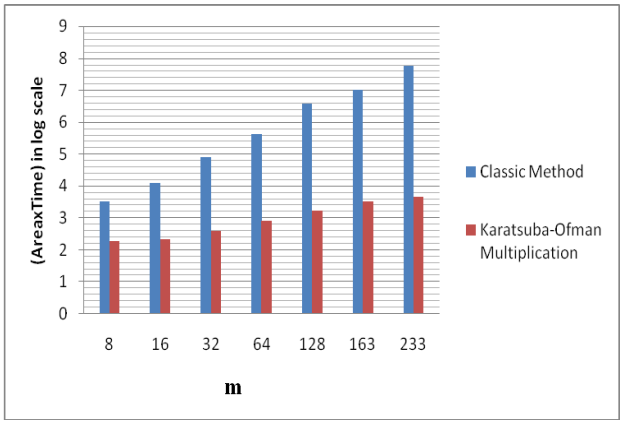
4. ANALYSIS OF POLYNOMIAL MULTIPLIER

The hardware design of the multiplier is desired to have the following properties:

1. Minimum *Area*×*Time (AT)*
2. Minimum *power*× *Delay*

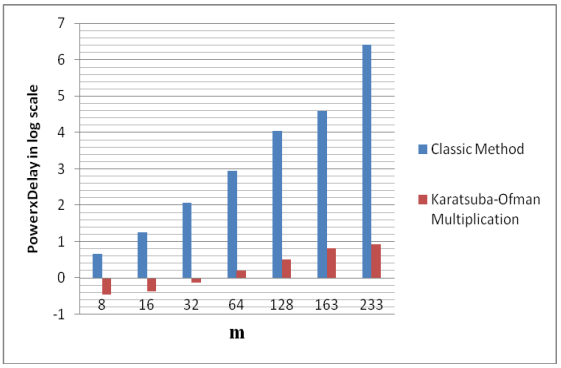
4.1 AT ANALYSIS

Graph-4.1 shows *Area*×*Time (AT)* characteristics of multiplication algorithms. This is one of important characteristics to analyze the performance of multiplication algorithm design. It can be inferred from the above graph that for small value of *m*, *AT* is highest for the Classic multiplier. Therefore, interleaved multiplication algorithm has minimum *AT* for all values of *m*.



Graph-4.1: (Area × Time) graph of Multiplier Algorithms for different *m*

4.2 Power × Delay ANALYSIS



Graph-4.2: Power × Delay Vs *m* for different algorithm

Another figure of merit for the design of a multiplier is the product of the power and delay values. *Power×Delay* graph is shown in the Graph-4.2. It shows that the designs based on Classic Multiplier algorithms have large values of the power-delay product, hence making them impractical for the design purpose. The design based on the interleaved multiplication algorithm has significantly low values of the product, making them viable options for design of a multiplier.

5. CONCLUSION

Performance analysis of two finite field multiplier algorithms has been done in terms of area, power and time. Further a comparative study between the performances of these algorithms has been done with the following inferences:

1. Synthesis results for the Classic Multiplier Algorithm show that total cell area, power and delay increase with *m* in orders greater than unity. It has large values of the power-delay product. It implies that hardware design of a multiplier based on the Classic Multiplier Algorithm is impractical for higher values of *m*.
2. For all values of *m*, small as well as large, the Karatsuba-Ofman multiplier yields best characteristics for area, power requirements and *Area×Time (AT)* characteristics. It has a significantly lower value of the power-delay product compared to Classic Multiplier. The design for Karatsuba-Ofman multiplier stands out to be best suitable for multiplier systems for cryptography, error-control coding and computer algebra

Based on the study of the performance of the design for each of the individual multiplier, and a comparative study between their performance parameters, it can be concluded that the *Karatsuba-Ofman multiplier* is the best option in two method for hardware design of an efficient multiplier. This design can be used as a multiplier in systems cryptography and error correcting codes.

REFERENCES

[1]. J.S. Milne, “Fields and Galois Theory”, Version 4.21, September, 2008.

[2]. Chen Guanghua, Liu Ming, Zhu Jingming and Zheng Weifeng, “Improvement of Interleaved Multiplication Algorithm”, the 1st International Conference on Information Science and Engineering (ICISE2009), 978-0-7695-3887-7/09, pp. 1752-1755

[3]. Peter Kornerup, “High-Radix Modular Multiplication for Cryptosystems”, IEEE 1063-6889/93, pp. 277 – 283

[4]. David Narh Amanor, Christof Paar, Jan Pelzl, Viktor Bunimov, Manfred Schimmler “Efficient Hardware Architectures for Modular Multiplication on FPGAS” 0-7803-9362-7/05, pp.539-542

[5]. Diego Viot, Rodolfo Aurélio, Helano Castro and Jardel Silveira, “Modular Multiplication Algorithm For Pkc”

[6]. H. Fan and M.A. Hasan, “Alternative to the Karatsuba algorithm for software implementations of GF(2n) multiplications”, IET Inf. Secur., 2009, Vol. 3, Iss. 2, pp. 60–65

[7]. Gang Zhou, Harald Michalik, and Laszlo Hinsenkamp, “Complexity Analysis and Efficient Implementations of Bit Parallel Finite Field Multipliers Based on Karatsuba-Ofman Algorithm on FPGAs”, IEEE Transactions On Very Large Scale Integration (VLSI) Systems, 1063-8210, pp.1-10.

[8]. El Hadj Youssef Wajih, Zeghid Medien, Machhout Mohsen, Bouallegue Belgacem and Tourki Rached “Efficient Hardware Architecture of Recursive Karatsuba-Ofman Multiplier”, 2008 International Conference on Design & Technology of Integrated Systems in Nanoscale Era, 978-1-4244-1577-9/08, pp.1-6.

[9]. Nadia Nedjah and Luiza de Macedo Mourelle, “A Reconfigurable Recursive and Efficient Hardware for Karatsuba-Ofman’s Multiplication Algorithm”, IEEE 2003, 0-7803-7729-X/03,pp.1076-1081

[10]. Serdar S. Erdem and C, etin K. Koc “A Less Recursive Variant of Karatsuba-Ofman Algorithm for Multiplying Operands of Size a Power of Two”, Proceedings of the 16th IEEE Symposium on Computer Arithmetic (ARITH'03), 1063-6889/03

[11]. Steffen Peter and Peter Langend¨orfer, “An Efficient Polynomial Multiplier in GF(2^m) and its Application to ECC Designs” EDAA-2007, 978-3-9810801-2-4

[12]. Viktor Bunimov and Prof. Dr. Manfred Schimmler, “Area and Time Efficient Modular Multiplication of Large Integers”, Proceedings of the Application-Specific Systems, Architectures, and Processors (ASAP’03) ISBN0-7695-1992-X/03 pp 1 -10.

[13]. E. Savas, A.F. Tenca, M.E. C¸ iftc¸ibasi and C¸ .K. Koc, “Multiplier architectures for GF (p) and GF (2^n)”, IEEE-2004 Comput. Digit. Tech., Vol. 151, No. 2,pp 147-160 March 2004

[14]. J. Bhasker, “A VHDL Primer”, Thrid Edition, Pearson Prentice Hall, 2008.

[15]. “Design Compiler Command-Line Interface Guide” Version 2006.06, synopsys.