An alternative approach to modulo-multiplication for Finite Fields using the Itoh-Tsujii Algorithm

BHARATHWAJ SANKARA VISWANATHAN [(1)],
KISHORE LAKSHMI NARASIMHAN [(2)].
Department of Electronics and Communications Engineering,
Sri Venkateswara College of Engineering,
Pennalur, Sriperumbudur – 602105.
INDIA

Abstract - Modulo arithmetic operations especially modulo multiplication have extensive applications in elliptic curve cryptanalysis, error control coding and linear recurring sequences. These operations have steadily grown in the word size in the past. Current designs and approaches may not be the most efficient for such high word sizes. Also usually, most approaches optimize for either area or speed, not both.

In this paper, we examine certain properties and elucidate certain alternative strategies of and on the Itoh-Tsujii algorithm[1] that will make it suitable for this emerging scenario. These strategies take a holistic approach to the problem, and aims at optimizing both speed and area for a given word length. These claims are supported by mathematical analysis, simulation and synthesis of a prototype of the suggested strategy. We also examine various enhancements that cam be effected in the given architecture.

Key Words – Modulo multiplier, high word size, Itoh Tsuji, scalability and pipelining.

1. Introduction

Modulo arithmetic units especially modulo multiplication operation plays an important role in a number of applications. These applications range from linear recurring sequences to elliptic curve cryptanalysis[2]. With the development of technology, the need for high bit size modulo multipliers became inevitable.

A number of present architectures give importance to either speed or area. But most of them fail to provide an optimization of both speed and area. In this paper we explore the possibilities of designing a modulo multiplication unit which can provide a good operating speed with sufficiently lesser area.

2. Mathematical Preliminaries

Several operations are defined at byte level, with bytes representing elements in the finite field GF(2^8). Other operations are defined in terms of 4-byte words. In this section we introduce the basic mathematical concepts needed in the following of the document.

2.1 The field GF(2^8)

The elements of a finite field can be represented in several different ways. For any prime power there is a single finite field, hence all representations of GF(2^8) are isomorphic. Despite this equivalence, the representation has an impact on the implementation complexity. We have chosen for the classical polynomial representation.

A byte b, consisting of bits $b_7\ b_6\ b_5\ b_4\ b_3\ b_2\ b_1\ b_0$, is considered as a polynomial with coefficient in {0,1}:

$$b7\ x^7 + b6\ x^6 + b5\ x^5 + b4\ x^4 + b3\ x^3 + b2\ x^2 + b1\ x^1 + b0 \quad \ldots(1)$$

Example: the byte with hexadecimal value '57' (binary 01010111) corresponds with polynomial $x^6 + x^4 + x^2 + x^1 + 1$.

2.1.1 Addition

In the polynomial representation, the sum of two elements is the polynomial with coefficients that are given by the sum modulo 2 (i.e., 1 + 1 = 0 (xor operation – denoted by "+")) of the coefficients of the two terms.

Example: '57' + '83' = 'D4', or with the polynomial notation:

$$(x^6 + x^4 + x^2 + x^1 + 1) + (x^7 + x^1 + 1) = x^7 + x^6 + x^4 + x^2 \quad \ldots(2)$$

In binary notation we have: "01010111" + "10000011" = "11010100". Clearly, the addition corresponds with the simple bit-wise EXOR (denoted by +) at the byte level.
All necessary conditions are fulfilled to have an Abelian group: internal, associative, neutral element ('00'), inverse element (every element is its own additive inverse) and commutative. As every element is its own additive inverse, subtraction and addition are the same.

2.1.2 Multiplication

In the polynomial representation, multiplication in GF (28) corresponds with multiplication of polynomials modulo an irreducible binary polynomial of degree 8. A polynomial is irreducible if it has no divisors other than 1 and itself. This polynomial is called m(x) and for GF(2^8) can be given as

$$m(x) = x^8 + x^4 + x^3 + x + 1 \quad \ldots(3)$$

Or '11B' in hexadecimal representation.

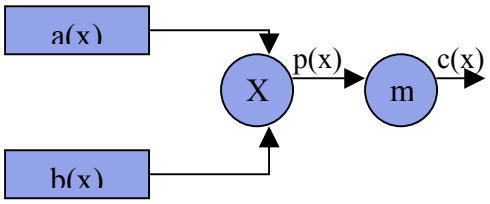

Fig 1: Block Diagram of modulo multiplier

Example: '57' · '83' = 'C1', or:

$$(x^6 + x^4 + x^2 + x^1 + 1)\,(x^7 + x^1 + 1)$$
$$= x^{13} + x^{11} + x^9 + x^8 + x^7 + x^7 + x^6 + x^3 + x^2 + x^1 + x^6 + x^4 + x^2 + x^1 + 1 \quad \ldots(4)$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^5 + x^4 + x^3 + 1$$

$$= x^{13} + x^{11} + x^9 + x^8 + x^6 + x^6 + x^4 + x^3 + 1 \text{ modulo } x^8 + x^4 + x^3 + x^1 + 1 \quad \ldots(5)$$

Hence,

$$(x^6 + x^4 + x^2 + x^1 + 1) * (x^7 + x^1 + 1) = x^7 + x^6 + 1 \quad \ldots(6)$$

Clearly, the result will be a binary polynomial of degree below 8. Unlike for addition, there is no simple operation at byte level. The multiplication defined above is associative and there is a neutral element ('01'). For any binary polynomial b(x) of degree below 8, the extended algorithm of Euclid can be used to compute polynomials a(x), c(x) such that b(x). a(x) + m(x). c(x) = 1. Hence, a(x) · b(x) mod m(x)= 1 or b-1(x) = a(x) mod m(x).
Moreover, it holds that

$$a(x) \cdot (b(x) + c(x)) = a(x) \cdot b(x) + a(x) \cdot c(x). \quad \ldots(7)$$

It follows that the set of 256 possible byte values, with the EXOR as addition and the multiplication defined as above has the structure of the finite field GF(2^8).

It is noted that the modulo operation is the most binding constraint in the implementation of the modulo arithmetic unit in a number of application[5].

3. Itoh-Tsujii Algorithm Based Multiplier[3]

Suppose input is y in GF(2^8) , y-1 has to be calculated. In GF(2^8) ,

$$y^{255} = 1$$

is satisfied then, $y^{-1} = y^{-1} * y^{255} = y^{254}$.
Consequently, y^{254} is equivalent to y^{-1}. Figure 1 shows an Inversion circuit based on Itoh and Tsujii's algorithm[4].

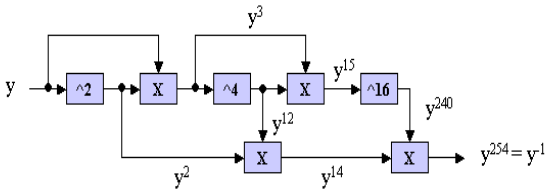

Fig 2: Inversion circuit based on Itoh-Tsujii

Example: Consider the multiplication of two 8-bit value A= (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0), B=(b_7, b_6,b_5,b_4, b_3, b_2, b_1, b_0). Since those 8 bit value can be expressed in polynomials in Galois Field such as

$$A(x) = a_7x^7+a_6x^6+a_5x^5+a_4x^4+a_3x^3+a_2x^2+a_1x+a_0 \quad \ldots(7)$$

$$B(x) = b_7x^7+b_6x^6+b_5x^5+b_4x^4+b_3x^3+b_2x^2+b_1x+b_0 \quad \ldots(8)$$

Then C(x) = A(x)*B(x) is given by

$$C(x) = c_{14}x^{14}+c_{13}x^{13}+c_{12}x^{12}+c_{11}x^{11}+c_{10}x^{10}+c_9x^9+c_8{}^8 +c_7x^7+c_6x^6+c_5x^5+c_4x^4+c_3x^3+c_2x^2+c_1x+c_0 \quad \ldots(9)$$

The result is 14th order polynomials. To bring the result back to the GF(2^8), perform the modulo operation using the irreducible polynomial. The Itoh-Tsujii algorithm simplifies the modulo operation as shown below:

$m(x)= x^8 + x^4 + x^3 + x + 1 = 0$ …(10)
$x^8 = x^4 + x^3 + x + 1$ …(11)
$x^9 = x^5 + x^4 + x^2 + x$ …(12)
$x^{10} = x^6 + x^5 + x^3 + x^2$ …(13)
$x^{11} = x^7 + x6 + x^4 + x^3$ …(14)
$x^{12} = x^8 + x^7 + x^5 + x^4 = x^7 + x^5 + x^3 + x + 1$ (15)
$x^{13} = x^8 + x^6 + x^4 + x^2 + x = x^6 + x^3 + x^2 + 1$ (16)
$x^{14} = x^7 + x^4 + x^3 + x$ …(17)

Now the result of the modulo operation is given below:

$D(x) = d_7x^7+d_6x^6+d_5x^5+d_4x^4+d_3x^3+d_2x^2+d_1x+d_0$ …(18)

$d_7=c_7 \oplus c_{12} \oplus c_{14}$ …(19)

$d_6=c_6 \oplus c_{10} \oplus c_{11} \oplus c_{13}$ …(20)

$d_5=c_5 \oplus c_9 \oplus c_{10} \oplus c_{12}$ …(21)

$d_4=c_4 \oplus c_8 \oplus c_9 \oplus c_{11} \oplus c_{14}$ …(22)

$d_3=c_3 \oplus c_8 \oplus c_{10} \oplus c_{11} \oplus c_{12} \oplus c_{13} \oplus c_{14}$ …(23)

$d_2=c_2 \oplus c_9 \oplus c_{10} \oplus c_{11}$ …(24)

$d_1=c_1 \oplus c_8 \oplus c_9 \oplus c_{12} \oplus c_{14}$ …(25)

$d_0=c_0 \oplus c_8 \oplus c_{12} \oplus c_{13}$ …(26)

Since D(x) expresses $GF(2^8)$, the multiplication result of two numbers in $GF(2^8)$ $A= (a_7, a_6, a_5, a_4, a_3, a_2, a_1, a_0)$ $B=(b_7, b_6, b_5, b_4, b_3, b_2, b_1, b_0)$ will be $D=(d_7, d_6, d_5, d_4, d_3, d_2, d_1, d_0)$.
In another words, multiplication in $GF(2^8)$ can be implemented using AND and EXOR logics.

3.1 Mathematical Analysis

Let the operands be in the $GF(2^m)$ field and let n be the order of multiplication. The results will have $\{d_{m-1}, \ldots. d_6, d_5, d_4, d_3, d_2, d_1, d_0\}$ as the coefficient. Further the work done depends on m(x). Let m(x) be $\{m_m, \ldots. m_6, m_5, m_4, m_3, m_2, m_1, m_0\}$.

In general,
$d_k = \Sigma m_j (\Sigma c_i)$ …(27)

where j = 0 to m-1
and i = k-j, m+k-j, 2m+k-j, …….

The computational complexity can be approximated to **O(n/m)**, which is a very good factor.

3.2 Synthesis report:

The synthesis of the Galois multiplier for word lengths 8, 16 and 32 using Xilinx Virtex600e are shown.

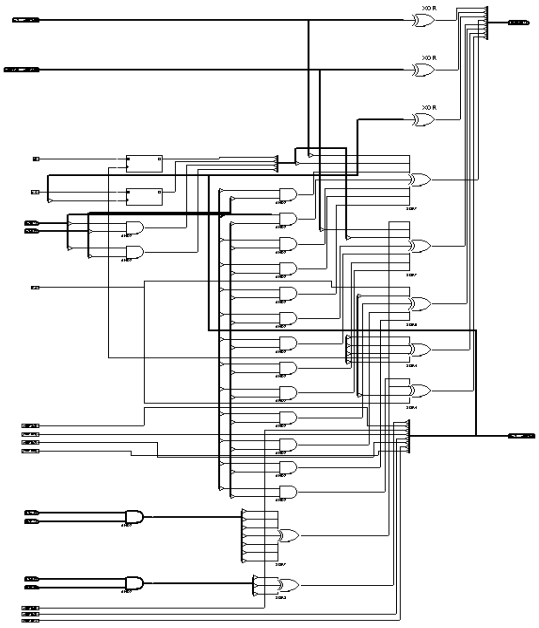

Fig 3: RTL Schematic of 8-bit GALOIS multiplier

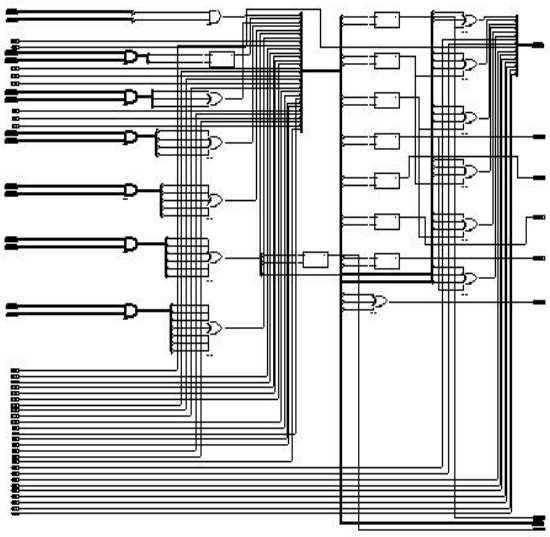

Fig 4: RTL-Schematic of a 16-bit GALOIS multiplier

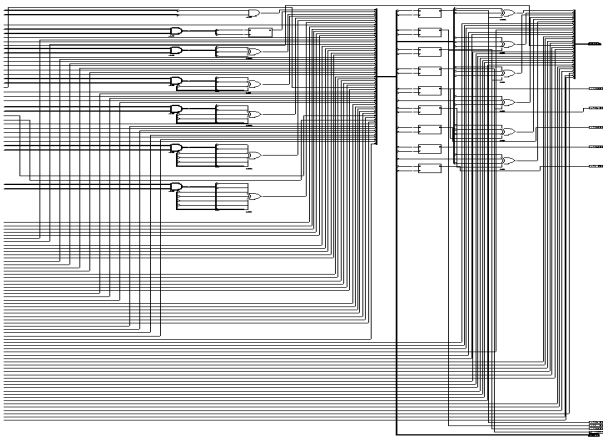

Fig 5: RTL Schematic of a 32-bit GALOIS multiplier

3.2.1 Resource Utilization Summary:

Number of bits	Number of slices	Number of 4 input LUTs	Number of Bonded IOBs
8	30	52	24
16	114	198	48
32	471	819	96

Table 1: Resource utilization

3.2.2 Timing Report:

Number of bits	Logic Delay (ns)	Route Delay (ns)	Maximum Combinational Path Delay (ns)
8	7.407	6.640	14.047
16	7.407	7.144	14.551
32	8.203	9.088	17.291

Table 2: Timing report

4. Suggested Enhancements:

First, the multiplier can be made scalable to take into consideration arbitrary values of the word length. This means the unit must be designed to compute the product in Galois field over a range of word lengths and different irreducible polynomial. The choice of the upper limit on the word size bears a constraint on both the speed and the area the multiplier occupies. Thus the maximum order 'm' of the GF(2^m) multiplier is decided based on the sacrifice that can be made on the speed. This also has to take into account the suitable irreducible polynomial of a given order that can be employed to cut down the number of IO buffers. An irreducible polynomial with a fewer non-zero coefficients will serve to reduce fan-in of the XOR gates.

Next, the multiplier can be enhanced using pipelining. Tomasulo`s model for pipelining can be exploited here. The computation of the product can be fragmented into simpler operations and incorporating pipeline registers for storing the result of the individual process.

A number of architectures have been proposed that concentrate either on the speed maximization [5][6] or on the area optimization [7]. Focusing on the aforesaid dependencies, an efficient Galois multiplier can be realized that will produce an optimal speed-area ratio.

5. Conclusion:

Itoh-Tsujii based Modulo-Multiplier is a good solution for application requiring high bit sizes. Further scalability and pipelining provides a multifold increase in its performance.

6. References

[1] J. Guajardo and C. Paar. *Itoh-Tsujii Inversion in Standard Basis and Its Application in Cryptography. Design, Codes, and Cryptography*, (25): 207–216, 2002

[2] Advanced Encryption Standard (AES), Federal Information Processing Standards Publication 197, November 26, 2001.

[3] [1] H. Brunner, A. Curiger, M. Hofstetter, *On computing multiplicative inverses in GF(2m),* IEEE Transactions on Computers, Vol. 42, No. 8, August 1993, pp. 1010-1015.

[4] T. Itoh and S. Tsujii. 'A Fast Algorithm for Computing Multiplicative Inverses in GF(2m) Using Normal Bases'. *Information and Computation*, 78: 171-177, 1988.

[5] E. D. Mastrovito. *VLSI Architectures for Computations in Galois Fields.* Dept. Electrical Engg., Linkoping, Sweden, 1991.

[6] J. L. Massey and J. K. Omura. *Computational method and apparatus for finite field arithmetic*. U.S. Patent Application, 1981.

[7] M. A. Hasan and V. K. Bhargava. Bit-serial systolic divider and multiplier for finite fields GF(2^m). *IEEE Transactions on Computers, 4*1(8): 972-980, Aug 1992.