The 2004 IEEE Asia-Pacific Conference on
Circuits and Systems, December 6-9, 2004

An Universal VLSI Architecture for Bit-Parallel Computation in GF(2^m)

Chien-Ching Lin, Fuh-Ke Chang, Hsie-Chia Chang, and Chen-Yi Lee

Department of Electronics Engineering
National Chiao Tung University
Hsinchu, Taiwan, R.O.C
Email: cclin@si2lab.org

ABSTRACT

In this paper, an universal VLSI architecture for bit-parallel computation in GF(2^m) is presented. The proposed architecture is based on Montgomery multiplication algorithm, which is suitable for multiple class of GF(2^m) with arbitrary field degree m. Due to the highly regular and modular property, our proposed universal architecture can meet VLSI design requirement.

After implemented by 0.18um 1P6M process, our universal architecture can work successfully at 125MHz clock rate. For the finite field multiplier, the total gate count is 1.4K for GF(2^m) with any irreducible polynomial of field degree m≤8, whereas the inverse operation can be achieved by the control unit with gate count of 0.3K. [1]

Table 1: Different finite field definitions for various applications, where p(x) is irreducible polynomial.

Cryptosystem		AES system for GF(2^8), $p(x) = x^8+x^4+x^3+x+1$
Flash		(520,512) RS code for GF(2^{10}), $p(x)= x^{10}+x^3+x^2+x^1+1$
ITU J.83	Annex B	(128,122) RS code for GF(2^7), $p(x) = x^7+x^3+1$
	Annex A,C	(204,188) RS code for GF(2^8), $p(x) = x^8+x^4+x^3+x^2+1$
	Annex D	(207,187) RS code for GF(2^8), $p(x) = x^8+x^4+x^3+x^2+1$
Blu-ray Disc	LDC	(248,216) RS code for GF(2^8), $p(x) = x^8+x^4+x^3+x^2+1$
	BIS	(62,30) RS code for GF(2^6), $p(x) = x^6+x^4+x^3+x^2+1$
	ADIP	(15,9) RS code for GF(2^4), $p(x) = x^4+x+1$

1. INTRODUCTION

The finite field computation is essential in various applications such as error correcting code and cryptosystem. However, there are multiple definitions of the finite fields for different applications, and the corresponding multipliers are slightly different due to the irreducible polynomials. The Table 1 shows several examples of different finite field application. Because the field degrees and irreducible polynomials are not matching, the multiplication operations must be executed by dedicated finite field multipliers leading to large circuit complexity.

With the rapidly increasing interesting in digital signal processing techniques, many systems utilize digital signal processor (DSP) to process data in real time and offer fast time to market. Thus, there is a demand for the universal finite field computation in software-based DSP implementation.

Actually, the finite field addition and subtraction are just excursive OR operations. Therefore, what we interest is the multiplication and division (or say, the inverse operation) in finite field. The complexity of finite field multiplication is in modular operation. Hence, the Montgomery algorithm can be used to simplify the modular operation. This paper describes this algorithm to accommodate different irreducible polynomials in multiple finite fields and also eliminate the effect of the field degree *m*.

First, in this paper, the Montgomery multiplication algorithm [1] is introduced in section 2. Section 3 describes an universal finite field multiplier with two stage Montgomery multipliers. In section 4, we introduce the Fermat algorithm and realize the finite field inverse operation in universal multiplier. Section 5 shows the implementation result of universal multiplier and inverter. Finally, the conclusion is given in section 6.

2. MONTGOMERY MULTIPLICATION ALGORITHM

An element A of the field GF(q^m) with a prime q can be interpreted as the polynomial representation:

$$A(x) = \sum_{i=0}^{m-1} a_i x^i = a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \ldots + a_1 x + a_0$$

,where $a_i \in GF(q)$

In the polynomial representation, multiplication in $GF(q^m)$ corresponds to the multiplication of polynomials module an irreducible polynomial of degree m. Suppose A and B are two elements in $GF(q^m)$, and $\mu(x)$ is the corresponding irreducible polynomial of this field. Then,

The authors are grateful to the support from the National Science Council of Taiwan, R.O.C., under the grant NSC 92-2218-E-009-019

the multiplicative operation C=AB can be expressed as follows:

$$C(x) = A(x)B(x) \mod \mu(x) \quad (2)$$

where C is also an element of GF(q^m). According to the modular multiplication property in (2), we can adopt Montgomery multiplication algorithm to calculate the product $C(x)$. The following equation defines the Montgomery product of A and B:

$$\hat{C}(x) = A(x)B(x)R^*(x) \mod \mu(x) \quad (3)$$

The polynomial $R^*(x)$ here is a fixed element of GF(q^m) satisfying $R(x)R^*(x) = 1 \ mod \ \mu(x)$ while $R(x)=x^m$. Note that the requirement of $R(x)$ and $\mu(x)$ being relatively prime is always consistent. It has been proven by [1] that the result $\hat{C}(x)$ of (3) can be obtained by following equations:

$$Q(x) = A(x)B(x)\mu^*(x) \mod R(x) \quad (4)$$

$$\hat{C}(x) = [A(x)B(x) + Q(x)\mu(x)] / R(x) \quad (5)$$

The polynomial $\mu^*(x)$ in (4) is defined as $\mu(x)\mu^*(x)=1 \ mod \ R(x)$. As compared with (3), it is evident that modulo μ(x) operation is replaced by modulo $R(x)$ and division by $R(x)$ operations. Since $R(x)=x^m$, implementation of (4) and (5) are much easier than that of (3). Furthermore, as A is interpreted in polynomial form and $R^*(x)= x^{-m} \ mod \ \mu(x)$, (3) can be rewritten as:

$$\hat{C}(x) = [a_{m-1}B(x)x^{-1} \bmod \mu(x)] + [a_{m-2}B(x)x^{-2} \bmod \mu(x)] + \ldots + [a_0B(x)x^{-m} \bmod \mu(x)]$$

Rearrange this equation, an iterative representation comes out:

$$\hat{C}(x) = [a_{m-1}B(x) + [\ldots[a_1B(x) + [a_0B(x)x^{-1} \bmod \mu(x)]]x^{-1} \bmod \mu(x)]\ldots]x^{-1} \bmod \mu(x)$$

Based on this equation and the transformation from (3) to (5), the Montgomery multiplication algorithm is derived as:

Montgomery multiplication algorithm:

$S_0(x) = 0;$
$for(i = 0; i < m; i++)\{$
$\quad \rho_i(x) = [(S_i(x) + a_iB(x))\mu^*(x)] \bmod x;$
$\quad S_{i+1}(x) = [S_i(x) + a_iB(x) + \rho_i(x)\mu(x)] / x;$
$\}$
$\hat{C}(x) = S_m(x);$

The term $\mu^*(x)$ is the multiplicative inverse of $\mu(x)$ under modulo x multiplication.

3. UNIVERSAL FINITE FIELD MULTIPLIER FOR GF(2^m)

In GF(2^m), elements are often represented in binary digits, and the coefficients a_i in (1) are referred to the bits of A. The binary representation will cause some reduction to Montgomery multiplication algorithm. Since $\mu(x)$ is irreducible, the results of μ(x) mod x and $\mu^*(x) \ mod \ x$ are both equal to 1. The $\mu^*(x)$ term in the Montgomery multiplication algorithm can be eliminated, which leads $\rho_i(x)$ to equal the least significant bit of the sum $S_i(x) + a_iB(x)$.

The number of recursive operation in Montgomery multiplication depends on the field degree m. However, some modification can be proposed to remove the effect of unexpected variable m. In equation (4) and (5), $R(x)$ is modified to be $R_d(x)=x^d$, and d is a constant integer such that $d \geq m$. Since the result of $R^*_d(x) \ mod \ \mu(x)$ is an element of GF(q^m), there exists an element $R^*_d(x)$ in the field GF(q^m) that satisfies $R_d(x)R^*_d(x)=1 \ mod \ x$. Therefore, the modified Montgomery multiplication (MM) algorithm for GF(2^m) with $m \leq d$ is constructed:

Modified Montgomery multiplication algorithm:

$MM(A(x), B(x), \mu(x))\{$
$\quad S_0(x) = 0;$
$\quad for(i = 0; i < d; i++)\{$
$\quad\quad if(i \geq m) \quad a_i = 0;$
$\quad\quad T(x) = S_i(x) + a_iB(x);$
$\quad\quad S_{i+1}(x) = [T(x) + t_0\mu(x)] / x;$
$\quad \}$
$\quad \hat{C}(x) = S_d(x);$
$\}$

The term t_0 is the least significant bit of the temporal element $T(x)$. If the field degree is less than d, the most significant bits of A is set to zero .The final result will be multiplying the normal finite field product $A(x)B(x)$ by a constant element $R^*_d(x)$ of GF(2^m). The output of Montgomery multiplier involves a constant factor $R^*_d(x) \ mod \ \mu(x)$ with the standard product. Such constant factor can be canceled by applying one additional Montgomery multiplier. Calculation of the product $C(x)=A(x)B(x)$ is completed using:

$$K(x) = x^{2d} \bmod \mu(x) \quad (8)$$

$$\hat{C}(x) = MM(A(x), B(x), \mu(x)) \quad (9)$$

$$C(x) = MM(\hat{C}(x), K(x), \mu(x)) \quad (10)$$

where $K(x)$ is treated as a constant value for a given $\mu(x)$.

The Montgomery multiplier architecture for GF(2^m) with $m \leq 4$ is shown in Fig.1. Fig.1(a) and Fig.1(b) indicate the function unit of Montgomery multiplier. Fig.1(c) shows the overall architecture of Montmogery multiplier *in GF(2^4)*. The symbol a_i and b_i are the bits of two input element A and B, which are often expressed as A=($a_3a_2a_1a_0$) and B=($b_3b_2b_1b_0$) respectively. Similarly, m_i is used to indicate the bits of irreducible polynomial and S_i are the output bits. Note that m_0 is always 1, and can be neglected after simplification. The proposed

architecture has a hardware complexity of $O(2d^2+d)$ and a latency of $O(d)$ while the numeric d is the maximum field degree of a field that the multiplier can deal with.

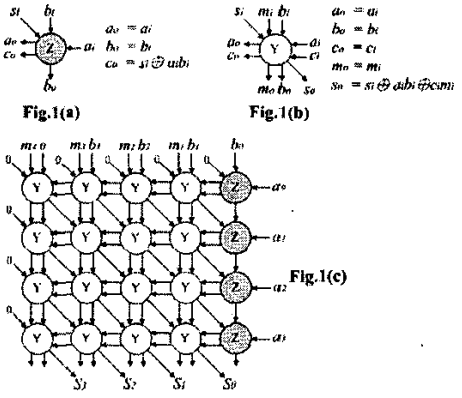


Fig.1(a) Fig.1(b) Fig.1(c)

Fig.1 ***Montgomery multiplier(MM) structure for GF(2^m) for $m \leq 4$***

As the standard multiplication requires two Montgomery multiplications, an overall standard multiplier demands two Montgomery multipliers connected in series as shown in Fig. 2. However, in many situations such as finite field inversion and multiply-and-add operations in coding theory [3], not all of the multiplications need to have multiply-by-K(x) operations as shown in (10).

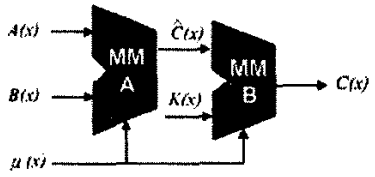


Fig.2: Two stage finite field multiplier

4. UNIVERSAL FINITE FIELD INVERTER FOR GF(2^M)

The finite field inverse operation is used frequently by many algorithms. Usually, the inverse operation is realized by looking up table, which costs much silicon area. Two-staged universal finite field multiplier is proposed to implement finite field inversion. The Fermat's algorithm [4] is described which can express the inversion as serial operations.

Fermat algorithm

$$\beta^{-1} = \beta^{2^m-2}$$

$$= \beta^{2+2^2+\ldots+2^{m-1}}$$

$$= \beta^{2(1+2(1+\ldots))}$$

$$= (\beta\ldots(\beta(\beta * \beta^2)^2)^2\ldots)^2$$

Based on this algorithm, the inversion in GF(2^m) can be replaced by serial square and multiply operations. In additional, the Fermat algorithm shows us that inverse operation needs $m-1$ cycles which include two Montgomery multiplications in each cycle. For example, in GF(16), $\beta^{-1} = \beta^{16-2} = \beta^{2+4+8} = \beta^{2(1+2(1+2))} = (\beta(\beta\beta^2)^2)^2$, three cycles are required.

Fig.3 shows the architecture of inverter which is almost the same with multiplier. A control unit is added to realize the inverse operation, and the Montgomery multiplier A (MMA) is taken as a squarer. Table 2 shows the register contents during inverse operation. Initially, we input the finite field element to MMA and MMB. Then, during 1 to $m-1$ cycle, the input value is the previous output. At cycle $m-1$, the input value $K'(x)$ of register k is a constant, where

$$K'(x) = x^{-d} \bmod \mu(x). \qquad (11)$$

We can know that $K(x)$ only varies with the finite field irreducible polynomial $\mu(x)$.

Base on Fermat algorithm, the finite field inversion is realized by using multiplications. In general, it takes $2m-3$ standard multiplications in computing the inverse of an element in $GF(2^m)$. If the Montgomery multiplier is adopted, it will result in $2m-2$ Montgomery multiplications. Another example is Berlekamp-Massey algorithm [3] in Reed-Solomon decoding procedure. The computation of discrepancy is a series of multiply-and-add operations [5], and only one extra multiplicative count is needed while using Montgomery multiplier.

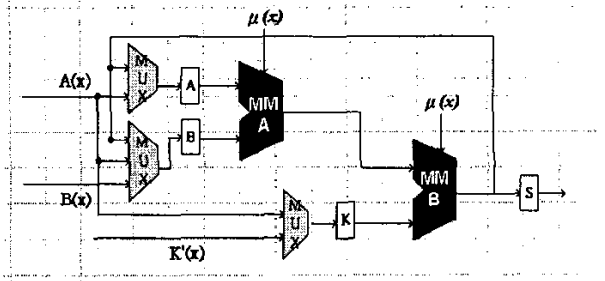


Fig.3: Universal finite field inverter

Table 2: Register state in inverse operation for GF(2^m)

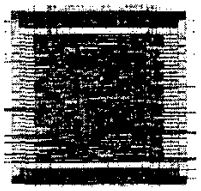
Register	Clock Cycle(two Montgomery multiplications)		
	0	1~m-3	m-2
Reg A	A(x)	Out of MMB	Out of MMB
Reg B	A(x)	Out of MMB	Out of MMB
Reg K	A(x)	A(x)	K'(x)
Reg S	0	0	Out of MMB

5. IMPLEMENTATION RESULTS

This implementation is designed by the standard cell library in a 0.18um 1P6M process. The layout area is 0.16·0.16 mm^2, and the total gate count is about 1.7K.

Only one cycle is required to compute finite field multiplication. The post layout simulation for the proposed architecture is about 125MHz. However, 250MHz can be achieved by inserting one pipeline register.

Technology:	.18um 1P6M
Core size:	0.16·0.16mm^2
Gate count:	1.7K
Speed:	125MHz



As compared to another finite field multiplier proposed by [6], our approach needs no additional pre and post-shifting. Table 3 compares the required instruction cycle between the proposed multiplier and the multiplier of [6] while operating over GF(2^m) with a multiplier that supporting maximum field degree of 8, where m is less than 8. Note that one instruction cycle here indicates a single shift operation, multiplication, or addition. And the finite field division in Table 3 is based on Fermat's algorithm.

Table 3: Comparison of universal multipliers

	Critical path	Instruction cycle		
		C=AB	C=A/B	$C=\sum_{i=0}^{n-1} A_i B_i$
L. Song [6]	$8T_{AND}$ $+11T_{XOR}$	3	$4m$-4	3n-2
Proposed	$9T_{AND}$ $+15T_{XOR}$	2	$2m$-1	$2n$

6. CONCLUSION

We proposed an universal multiplier for GF(2^m) based on Montgomery multiplication algorithm. As the multiplier for maximum field degree d is implemented, any multiplier for GF(2^m) with field degree less than d can be executed. The proposed bit-parallel multiplier is regular without additional control circuit. In a software based design of error correcting code or cryptosystem on a DSP, the universal finite field multiplier is useful because of variant GF(2^m) definition for different applications. Furthermore, the inverse operation can be realized by appending additional control unit to universal multiplier. This control unit only increases some gate count than original circuit. For each new specification, our approach requiring little modification can reduce not only the time to market but also the development cost.

7. REFERENCES

[1] C.K. Koc and T Acar, "Fast Software Exponentiation in GF(2^k)", 13th IEEE Symposium on Computer Arithmetic, pp. 225-231, 1997.

[2] R.J. McEliece, "Finite Fields for Computer Scientists and Engineers", Kluwer Academic Publishers, Boston, MA, 1987.

[3] R.E. Blahut, "Theory and Practice of Error Control Codes", Addison-Wesley Pulishing Company, MA, 1983.

[4] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in GF(2^m) Using Normal Bases", Journal of Information and Computation J., vol 78, pp.171-177, 1988.

[5] H.C. Chang, C.B. Shung and C.Y. Lee, "A Reed–Solomon Product-Code (RS-PC) Decoder Chip for DVD Applications", IEEE Joural of Solid-State Circuits, Vol. 1, No. 2, pp. 229-236, Feb. 2001

[6] L. Song, K.K. Parhi, I. Kuroda, and T. Nishitani, "Hardware/Software Codesign of Finite Field Datapath for Low Energy Reed-Solomon codecs", IEEE Transactions on Very Large Scale Integration Systems, Vol. 8, No. 2, pp. 160-172, Apr. 2000