Montgomery Multiplier and Squarer in GF(2^m)

Huapeng Wu

The Centre for Applied Cryptographic Research
Department of Combinatorics and Optimization
University of Waterloo, Waterloo, Canada
h3wu@cacr.math.uwaterloo.ca

Abstract. Montgomery multiplication in GF(2^m) is defined by $a(x)b(x)r^{-1}(x) \bmod f(x)$, where the field is generated by irreducible polynomial $f(x)$, $a(x)$ and $b(x)$ are two field elements in GF(2^m), and $r(x)$ is a fixed field element in GF(2^m). In this paper, first we present a generalized Montgomery multiplication algorithm in GF(2^m). Then by choosing $r(x)$ according to $f(x)$, we show that efficient architecture for bit-parallel Montgomery multiplier and squarer can be obtained for the fields generated with irreducible trinomials. Complexities in terms of gate counts and time propagation delay of the circuits are investigated and found to be comparable to or better than that of polynomial basis or weakly dual basis multiplier for the same class of fields.

1 Introduction

Finite field has applications in combinatorial designs, sequences, error-control codes, and cryptography. Finite field arithmetic operations have been paid much attention recently mainly because its use in cryptography, especially in elliptic curve cryptosystems. Research in this area has been characterized by its strong flavor of implementation both in software and in hardware. For example, fields of characteristic two are prevailingly used because a ground field operation can be readily implemented with a VLSI gate.[1]

In this paper, we first give a generalized Montgomery multiplication algorithm in GF(2^m). Then by choosing the fixed field element $r(x)$ according to the irreducible polynomial, we show that efficient multiplication and squaring architectures can be obtained using the generalized algorithm of Montgomery multiplication in GF(2^m). The implementation complexities in terms of the number of gates (equivalent to the number of ground field operations) and time propagation delay are lower than or as good as these of previously proposed multipliers for the same class of fields. The main implementation results are summarized in the two theorems.

[1] A multiplication operation in GF(2) can be implemented using an AND gate, while an addition operation in GF(2) can be implemented with an XOR gate.

Ç.K. Koç and C. Paar (Eds.): CHES 2000, LNCS 1965, pp. 264–276, 2000.

2 Preliminaries

Montgomery multiplication was first proposed for integer modular multiplication that can avoid trial division [2]. Later it was extended to finite field multiplication in GF(2^m) [1]. It was shown that the operation can be made simple if certain type of $r(x)$ is selected [1]. In the following, we give a brief review of the Montgomery multiplication in GF(2^m) proposed in [1].

Let $f(x)$ be the irreducible polynomial that defines the field GF(2^m) and $r(x)$ be a fixed element in GF(2^m). Since $\gcd(f(x), r(x)) = 1$, we can use the extended Euclidean algorithm to determine $f'(x)$ and $r'(x)$ that satisfy

$$r(x)r'(x) + f(x)f'(x) = 1. \tag{1}$$

Clearly $r'(x) = r^{-1}(x)$ is the inverse of $r(x)$. Given two field elements $a(x), b(x) \in$ GF(2^m), then an analogue for Montgomery multiplication in GF(2^m) can be given by [1]

$$c(x) = a(x)b(x)r^{-1}(x) \bmod f(x), \tag{2}$$

and an algorithm to compute (2) is shown below:

Algorithm 1. Montgomery multiplication in GF(2^m) [1]
Input: $a(x), b(x), r(x), f'(x)$
Output: $c(x) = a(x)b(x)r^{-1}(x) \bmod f(x)$
Step 1. $t(x) \Leftarrow a(x)b(x)$
Step 2. $u(x) \Leftarrow t(x)f'(x) \bmod r(x)$
Step 3. $c(x) \Leftarrow [t(x) + u(x)f(x)]/r(x)$

The correctness of Algorithm 1 can be easily checked. Note that

$$\begin{aligned} \deg[c(x)] &\leqslant \max\{\deg[t(x)], \deg[u(x)] + \deg[f(x)]\} - \deg[r(x)] \\ &= \max\{2m-2, \deg[r(x)] - 1 + m\} - \deg[r(x)] \\ &= \max\{2m-2-\deg[r(x)], m-1\}. \end{aligned}$$

Thus, to have $\deg[c(x)] \leqslant m-1$, the degree of $r(x)$ must be chosen not less than $m-1$. Since $f(x)$ and $f'(x)$ can be considered as constants, it is noted in [1] that efficient multiplication can be achieved if $r(x)$ is properly chosen. In fact, $r(x)$ was chosen to be the monomial x^m in [1] and Algorithm 1 is equivalent to a polynomial multiplication, two constant multiplications in GF(2^m) and one addition in GF(2^m).

3 Generalized Montgomery Multiplication in GF(2^m)

For bit-parallel realization of Montgomery multiplication in GF(2^m), we find that efficient multiplier architecture can be obtained if $r(x)$ is chosen according to the irreducible polynomial $f(x)$. For example, if the field is generated with a trinomial $f(x) = x^m + x^k + 1$, then $r(x)$ is selected to be the term of the second low degree in the trinomial. This choice of $r(x) = x^k$ turns out to be very helpful in obtaining low complexity multiplier and squarer architectures. However, Algorithm 1 can not directly be used for these cases since k can be less than $m-1$. This leads us to consider a generalized form of Montgomery multiplication in GF(2^m). In the following, we first present a generalized Montgomery algorithm in GF(2^m), then compare it with Algorithm 1.

Algorithm 2. Generalized Montgomery multiplication in GF(2^m)
Input: $a(x), b(x), r(x), f(x), f'(x)$
Output: $c(x) = a(x)b(x)r^{-1}(x) \bmod f(x)$
Step 1. $t(x) \Leftarrow a(x)b(x)$
Step 2. $u(x) \Leftarrow t(x)f'(x) \bmod r(x)$
Step 3. $\tilde{c}(x) \Leftarrow [t(x) + u(x)f(x)]/r(x)$
Step 4. If $\deg(\tilde{c}) > m-1$, then $c(x) \Leftarrow \tilde{c}(x) \bmod f(x)$, else $c(x) \Leftarrow \tilde{c}(x)$

The correctness check for the algorithm is similar to that of Algorithm 1.

The degree range of $\tilde{c}(x)$ can be estimated. Since $0 \leqslant \deg[a(x)] \leqslant m-1$ and $0 \leqslant \deg[b(x)] \leqslant m-1$, it follows $0 \leqslant \deg[t(x)] \leqslant 2m-2$. Assume $\deg[r(x)] = k$, then from Step 2 we have $0 \leqslant \deg[u(x)] \leqslant k-1$. From Step 3, we have

$$\deg[\tilde{c}(x)] \leqslant \max\{2m-k-2, m-1\}. \tag{3}$$

When $\deg[r(x)] = k < m-1$, the degree of $\tilde{c}(x)$ is $2m-k-2$ and higher than $m-1$. In this case, one more step of modulo reduction (Step 4) is needed.

Compared to Algorithm 1, Algorithm 2 extends the degree range that $r(x)$ can be chosen from. Algorithm 1 can be considered as a specific case of Algorithm 2. For example, when $r(x)$ is chosen such that $\deg r(x) \geqslant m-1$, then $\tilde{c}(x)$ obtained at Step 3 in Algorithm 2 has a degree equal to or less than $m-1$. In this case, Step 4 will not be performed and the algorithm is the same as Algorithm 1. In fact, Algorithm 2 looks more similar to the original Montgomery algorithm [2] than Algorithm 1. This is because Step 4 in Algorithm 2 corresponds to the final subtraction step in the original Montgomery algorithm [2]. In Algorithm 1 this step has been omitted provided that some condition has been applied to how to choose $r(x)$.

4 Montgomery Multiplier in GF(2^m)

Consider the irreducible polynomial $f(x) = x^m + x^k + 1$, $\frac{m}{2} \leqslant k \leqslant m-1$, and the fixed field element $r(x) = x^k$ for the Montgomery multiplication in GF(2^m) (Algorithm 2). From the Extended Euclidean Algorithm, we obtain $r^{-1}(x) = 1 + x^{m-k}$ and $f'(x) = 1$ that satisfy

$$r(x)r^{-1}(x) + f(x)f'(x) = 1.$$

To solve the coefficients of the product $c(x)$ in terms of these of $a(x)$ and $b(x)$ and thus to find efficient multiplier architectures, we proceed with each step of Algorithm 2 as follows.

4.1 Step 1 in Algorithm 2

Let polynomial basis representations of $a(x)$ and $b(x)$ be given by $a(x) = \sum_{i=0}^{m-1} a_i x^i,\ a_i \in \mathrm{GF}(2)$, and $b(x) = \sum_{i=0}^{m-1} b_i x^i,\ b_i \in \mathrm{GF}(2)$, respectively. Then $t(x) = a(x)b(x)$ can be obtained as follows;

$$t(x) = a(x)b(x) = \sum_{i=0}^{2m-2} t_i x^i, \tag{4}$$

where t_i's are given by

$$t_i = \begin{cases} \sum_{j=0}^{i} a_j b_{i-j}, & 0 \leqslant i \leqslant m-1, \\ \sum_{j=i-m+1}^{m-1} a_j b_{i-j}, & m \leqslant i \leqslant 2m-2. \end{cases} \tag{5}$$

It can be seen that total m^2 bit multiplications and $(m-1)^2$ bit additions in GF(2) are required to compute $t_i,\ i = 0, 1, \ldots, 2m-2$. An implementation of (5) is straightforward, and the gate counts and time delays incurred with signals $t_i,\ i = 0, 1, \ldots, 2m-2$, are listed in Table 1. We denote the time delays of an AND gate and an XOR gate by T_A and T_X, respectively.

Table 1. Complexity and Time Delay Involved in Implementing $t(x)$.

Signal	# AND gates	# XOR gates	Time delay
$t_0 = a_0 b_0$	1	0	T_A
$t_1 = a_0 b_1 + a_1 b_0$	2	1	$T_A + T_X$
$t_2 = a_0 b_2 + a_1 b_1 + a_2 b_0$	3	2	$T_A + 2T_X$
$t_3 = a_0 b_3 + a_1 b_2 + a_2 b_1 + a_3 b_0$	4	3	$T_A + 2T_X$
$\vdots = \vdots$	$\vdots$	$\vdots$	$\vdots$
$t_{m-2} = a_0 b_{m-2} + \cdots + a_{m-2} b_0$	$m-1$	$m-2$	$T_A + \lceil \log_2 (m-1) \rceil T_X$
$t_{m-1} = a_0 b_{m-1} + \cdots + a_{m-1} b_0$	m	$m-1$	$T_A + \lceil \log_2 m \rceil T_X$
$t_m = a_1 b_{m-1} + \cdots + a_{m-1} b_1$	$m-1$	$m-2$	$T_A + \lceil \log_2 (m-1) \rceil T_X$
$\vdots = \vdots$	$\vdots$	$\vdots$	$\vdots$
$t_{2m-3} = a_{m-2} b_{m-1} + a_{m-1} b_{m-2}$	2	1	$T_A + T_X$
$t_{2m-2} = a_{m-1} b_{m-1}$	1	0	T_A
Total:	m^2	$(m-1)^2$	$T_A + \lceil \log_2 m \rceil T_X$

In the following, we will solve the rest three steps of Algorithm 2 and show that they can be realized at one single implementation step.

4.2 Step 2 in Algorithm 2

Substitute $t(x)$ in this step using (4)

$$\begin{aligned} u(x) &= t(x)f'(x) \bmod r(x) \\ &= t_0 + t_1x + t_2x^2 + \cdots + t_{2m-2}x^{2m-2} \bmod x^k \\ &= t_0 + t_1x + t_2x^2 + \cdots + t_{k-1}x^{k-1}. \end{aligned} \tag{6}$$

Clearly, the degree of $u(x)$ is not higher than that of $r(x)$. If $r(x)$ is chosen to have a low degree then we have a simple $u(x)$.

4.3 Step 3 in Algorithm 2

Define

$$t_L(x) \stackrel{\triangle}{=} t_0 + t_1x + t_2x^2 + \cdots + t_{k-1}x^{k-1}, \tag{7}$$

and

$$t_H(x) \stackrel{\triangle}{=} t_k + t_{k+1}x + \cdots + t_{2m-2}x^{2m-k-2}. \tag{8}$$

From (4) (7) and (8), it can be seen that

$$t(x) = t_L(x) + x^k t_H(x), \tag{9}$$

and from (6) and (7) it follows

$$u(x) = t_L(x). \tag{10}$$

Substitute $t(x)$ and $u(x)$ in Step 3 with (9) and (10), respectively, and note that $f(x) = x^m + x^k + 1$, we have

$$\begin{aligned} \tilde{c}(x) &= [t(x) + u(x)f(x)]/r(x) \\ &= [t_L(x) + x^k t_H(x) + t_L(x)(x^m + x^k + 1)]/x^k \\ &= [x^k t_H(x) + x^k(x^{m-k} + 1)t_L(x)]/x^k \\ &= t_H(x) + x^{m-k}t_L(x) + t_L(x). \end{aligned} \tag{11}$$

When $k = m - 1$, from (3) we have $\deg \tilde{c}(x) \leqslant m - 1$. Clearly, in this case the degree of $\tilde{c}(x)$ has already been reduced to the proper range and Step 4 in Algorithm 2 is not necessary.

Extend each of the three terms at the right hand side of (11) for the case of $k = m - 1$, and from (7) and (8) we have

$$\begin{aligned} t_H(x) &= t_{m-1} + t_m x + \cdots + t_{2m-2}x^{m-1}, \\ xt_L(x) &= t_0x + t_1x^2 + \cdots + t_{m-2}x^{m-1}, \\ t_L(x) &= t_0 + t_1x + \cdots + t_{m-2}x^{m-2}. \end{aligned} \tag{12}$$

Then by comparing (12) with (11), and note $\tilde{c}(x) = c(x) = \sum_{i=0}^{m-1} c_i x^i$, we can write c_i as follows

$$
\begin{aligned}
c_0 &= t_0 + t_{m-1}, \\
c_1 &= t_0 + t_1 + t_m, \\
c_2 &= t_1 + t_2 + t_{m+1}, \\
&\vdots \\
c_{m-2} &= t_{m-3} + t_{m-2} + t_{2m-3}, \\
c_{m-1} &= t_{m-2} + t_{2m-2}.
\end{aligned}
$$

Rewrite the above expressions as

$$
c_i = \begin{cases} t_0 + t_{m-1}, & i = 0, \\ t_{i-1} + t_i + t_{m-1+i}, & i = 1, 2, \ldots, m-2, \\ t_{m-2} + t_{2m-2}, & i = m-1. \end{cases} \tag{13}
$$

It can be seen from (13) that each c_i can be obtained with 2 bit additions in GF(2), except that c_0 and c_{m-1} require one bit operation each. Thus, a bit-parallel realization of (13) needs $2m - 2$ XOR gates. Since the maximal number of terms on the right hand side of each equation in (13) is three, the maximal time propagation delay is $2T_X$.

When $\frac{m}{2} \leqslant k < m - 1$, from (3) we have $\deg \tilde{c}(x) > m - 1$. In this case, a step of modulo reduction is still needed.

4.4 Step 4 in Algorithm 2

From (8) we divide $t_H(x)$ into two parts: $t_H(x) = t_H^{(1)}(x) + t_H^{(2)}(x)$, where

$$
t_H^{(1)}(x) \triangleq t_k + t_{k+1}x + \cdots + t_{k+m-1}x^{m-1}, \tag{14}
$$

and

$$
t_H^{(2)}(x) \triangleq t_{k+m}x^m + t_{k+m+1}x^{m+1} + \cdots + t_{2m-2}x^{2m-k-2}. \tag{15}
$$

Substitute $t_H(x)$ in (11) with $t_H^{(1)}(x) + t_H^{(2)}(x)$ and note that $c(x) = \tilde{c}(x) \bmod f(x)$, we have

$$
\begin{aligned}
c(x) &= \tilde{c}(x) \bmod f(x) \\
&= [t_H^{(1)}(x) + t_H^{(2)}(x) + x^{m-k}t_L(x) + t_L(x)] \bmod f(x) \\
&= t_H^{(1)}(x) + x^{m-k}t_L(x) + t_L(x) + [t_H^{(2)}(x)] \bmod f(x). \qquad (16)
\end{aligned}
$$

Apply the modulo operation to each term on the right hand side of (15), it follows

$$\begin{aligned} t_{k+m}x^m \bmod f(x) &= t_{k+m}(1+x^k), \\ t_{k+m+1}x^{m+1} \bmod f(x) &= t_{k+m+1}(x+x^{k+1}), \\ &\vdots \\ t_{2m-2}x^{2m-k-2} \bmod f(x) &= t_{2m-2}(x^{m-k-2}+x^{m-2}). \end{aligned}$$

Adding the above $m-k-1$ equations together, we obtain

$$t_H^{(2)}(x) \bmod f(x) = \sum_{i=0}^{m-k-2} t_{m+k+i}x^i + \sum_{i=k}^{m-2} t_{m+i}x^i.$$

Split $t_H^{(2)}(x)$ into two parts:

$$t_H^{(2,1)}(x) \bmod f(x) \triangleq \sum_{i=0}^{m-k-2} t_{m+k+i}x^i, \tag{17}$$

$$t_H^{(2,2)}(x) \bmod f(x) \triangleq \sum_{i=k}^{m-2} t_{m+i}x^i. \tag{18}$$

Substitute $t_H^{(2)}(x)$ with $t_H^{(2,1)}(x)+t_H^{(2,2)}(x)$ in (16) and note that $c(x) = \sum_{i=0}^{m-1} c_i x^i$, it follows

$$\sum_{i=0}^{m-1} c_i x^i = t_L(x) + x^{m-k}t_L(x) + t_H^{(1)}(x) + t_H^{(2,1)}(x) + t_H^{(2,2)}(x). \tag{19}$$

Rewrite the equations (7), (14), (17), (18) and extend the term $x^{m-k}t_L(x)$ using (7), we have the following five equations for the five terms on the right hand side of (19), respectively:

$$\begin{array}{lll} (a) & t_L(x) = t_0 + t_1x + t_2x^2 + \cdots + t_{k-1}x^{k-1} & [0, k-1] \\ (b) & x^{m-k}t_L(x) = t_0x^{m-k} + t_1x^{m-k+1} + \cdots + t_{k-1}x^{m-1} & [m-k, m-1] \\ (c) & t_H^{(1)}(x) = t_k + t_{k+1}x + \cdots + t_{k+m-1}x^{m-1} & [0, m-1] \\ (d) & t_H^{(2,1)}(x) = t_{m+k} + t_{m+k+1}x + \cdots + t_{2m-2}x^{m-k-2} & [0, m-k-2] \\ (e) & t_H^{(2,2)}(x) = t_{m+k}x^k + t_{m+k+1}x^{k+1} + \cdots + t_{2m-2}x^{m-2} & [k, m-2] \end{array} \tag{20}$$

The last column in the above array is the degree range of the terms on the right-hand side of each equation. Now we are ready to solve the coefficients c_i by comparing (19) with (20).

In the following, we consider three cases:

Case 1: If $\frac{m+1}{2} < k < m-1$. We have $m-k-2 < m-k < k-1 < k$. By comparing (19) with (20), we can solve c_i's (the coefficient of the term x^i in $c(x)$). When $0 \leqslant i \leqslant m-k-2$, it can be seen from (20) that c_i takes on the terms from equations $(a), (c)$ and (d). When $i = m-k-1$, c_{m-k-1} has only two terms, one is from equation (a) and the other from (c). When i runs through from $m-k$ to $k-1$, c_i picks up the terms from equations $(a), (b)$ and (c). When $k \leqslant i \leqslant m-2$, c_i has three terms: one from equation (b), one from (c) and the other from (e). Finally, c_{m-1} has two terms from equations (b) and (c), respectively. We can write c_i's as follows

$$
\begin{array}{rlllll}
 & (a) & (b) & (c) & (d) & (e) \\
c_0 = & t_0 & & +t_k & +t_{k+m} & \\
c_1 = & t_1 & & +t_{k+1} & +t_{k+m+1} & \\
\vdots & \vdots & & \vdots & \vdots & \\
c_{m-k-2} = & t_{m-k-2} & & +t_{m-2} & +t_{2m-2} & \\
c_{m-k-1} = & t_{m-k-1} & & +t_{m-1} & & \\
c_{m-k} = & t_{m-k} & +t_0 & +t_m & & \\
\vdots & \vdots & \vdots & \vdots & & \\
c_{k-1} = & t_{k-1} & +t_{2k-m-1} & +t_{2k-1} & & \\
c_k = & & +t_{2k-m} & +t_{2k} & & +t_{k+m} \\
\vdots & & \vdots & \vdots & & \vdots \\
c_{m-2} = & & +t_{k-2} & +t_{m+k-2} & & +t_{2m-2} \\
c_{m-1} = & & +t_{k-1} & +t_{m+k-1} & &
\end{array}
\tag{21}
$$

where all the terms at the column (a), (b), ... are from the equations (a), (b), ... in (20), respectively. Now we can estimate the complexity to obtain the coefficients of the product from t_i's. From (21), it can be seen that $2m-2$ bit addition in GF(2) are used to solve c_i's. The longest time delay to generate c_i from t_i is $2T_X$.

Case 2. If $k = \frac{m+1}{2}$. We have $m-k-2 < m-k = k-1 < k$. By comparing (19) to (20) the coefficients of $c(x)$ can be written as follows

$$
\begin{array}{rlllll}
 & (a) & (b) & (c) & (d) & (e) \\
c_0 = & t_0 & & +t_k & +t_{k+m} & \\
c_1 = & t_1 & & +t_{k+1} & +t_{k+m+1} & \\
\vdots & \vdots & & \vdots & \vdots & \\
c_{m-k-2} = & t_{k-3} & & +t_{2k-3} & +t_{2m-2} & \\
c_{m-k-1} = & t_{k-2} & & +t_{2k-2} & & \\
c_{m-k} = & t_{k-1} & +t_0 & +t_{2k-1} & & \\
c_k = & & +t_1 & +t_{2k} & & t_{k+m} \\
\vdots & & \vdots & \vdots & & \vdots \\
c_{m-2} = & & +t_{k-2} & +t_{m+k-2} & & +t_{2m-2} \\
c_{m-1} = & & +t_{k-1} & +t_{m+k-1} & &
\end{array}
\tag{22}
$$

It can be seen that a realization of the above expressions requires $2m-2$ ground field operations. Since the most terms to sum up for each c_i is three, the maximal time delay is $2T_X$.

Case 3. If $k = \frac{m}{2}$. We have $m-k-2 = k-2 < k-1 < m-k = k$. The coefficients of the Montgomery product can be obtained from (19) and (20) as follows:

$$
\begin{array}{rllllll}
 & & (c) & (a) & (d) & (b) & (e) \\
c_0 & = & t_k & +(t_0 & +t_{k+m}) & & \\
c_1 & = & t_{k+1} & +(t_1 & +t_{k+m+1}) & & \\
\vdots & \vdots & & \vdots & \vdots & & \\
c_{k-2} & = & t_{m-2} & +(t_{k-2} & +t_{2m-2}) & & \\
c_{k-1} & = & t_{m-1} & +t_{k-1} & & & \\
c_k & = & t_m & & & +(t_0 & +t_{k+m}) \\
c_{k+1} & = & t_{m+1} & & & +(t_1 & +t_{k+m+1}) \\
\vdots & \vdots & & & & \vdots & \vdots \\
c_{m-2} & = & t_{m+k-2} & & & +(t_{k-2} & +t_{2m-2}) \\
c_{m-1} & = & t_{m+k-1} & & & +t_{k-1} &
\end{array} \tag{23}
$$

Note that the resued partial sums are put in the brackets. Then it can be seen from (23) that $2m-2-(k-1) = \frac{3}{2}m - 1$ bit additions in GF(2) are required to compute $c_0, \ldots, c_{m-1}$ from $t_0, \ldots, t_{2m-2}$. The time delay incurred here is still $2T_X$.

4.5 Bit-Parallel Multiplier Architecture

From the above discussion, it can be seen that a bit-parallel Montgomery multiplication in GF(2^m) is decided by (5) and one of the expressions (21), (22) and (23). A diagram for the bit-parallel multiplier architecture is shown in Fig. 1. The upper two modules (one all-AND-gate circuits and one all-XOR-gate circuits) are used to perform polynomial multiplication (Step 1 in Algorithm 2), while the module at the bottom (all-XOR-gate circuits) corresponds to the implementation of Steps 2 to 4 in Algorithm 2.

It can be seen from Table 1 that m^2 AND agtes and $(m-1)^2$ XOR gates are required for generating t_i. Then the coefficients of $c(x)$ can be generated from t_i using one of (21), (22) and (23). Obviously, the total number of gates required are

$$
\begin{array}{rl}
m^2 & \text{AND gates,} \\
m^2 - 1 & \text{XOR gates,}
\end{array}
$$

if the irreducible trinomial is $f(x) = x^m + x^k + 1,\ \frac{m}{2} < k \leqslant m-1$.

When $f(x) = x^m + x^{\frac{m}{2}} + 1$, the complexity is only

$$
\begin{array}{rl}
m^2 & \text{AND gates,} \\
m^2 - \frac{m}{2} & \text{XOR gates.}
\end{array}
$$

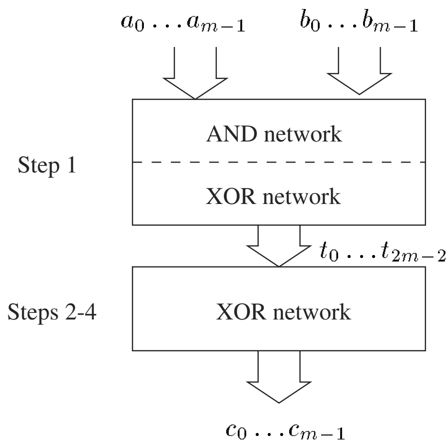

Fig. 1. Bit-Parallel Montgomery Multiplier Architecture when $f(x) = x^m + x^k + 1$ and $r(x) = x^k$.

Total time delay of the multiplier is not greater than $T_A + (\lceil \log_2 m \rceil + 2)\, T_X$. In many cases the total propagation delay is less than the above bound. Note from the Table 1 that the time delay incurred with different t_i is different. In fact, circuits for generating t_i has a time delay $\lceil \log_2(i+1) \rceil T_X$ if $i \leqslant m-1$, and $\lceil \log_2(2m-i-1) \rceil T_X$ if $i \geqslant m$. From (13), (21), (22) and (23), it can be seen that most c_i's is a sum of three terms. Write them as $c_i = t_{i1} + t_{i2} + t_{i3}$, where we assume that the time delays for generating t_{i1}, t_{i2}, and t_{i3} are d_1, d_2, and d_3, respectively. If $d_1 \leqslant d_2 \leqslant d_3$, then it can be seen that the propagation delay for generating c_i depends on d_2 and d_3 if the circuit is designed using

$$c_i = (t_{i1} \oplus t_{i2}) \oplus t_{i3}.$$

The time delay incurred with the above logic equation for generating c_i is

$$T_{c_i} = \max\{d_2 + 2, d_3 + 1\}.$$

Using this method, we search and find the maximal time delays incurred with the expressions (13), (21), (22) and (23).

4.6 Complexity Results and Example

We summarize the implementation results on Montgomery multiplier in GF(2^m) as follows:

Theorem 1. *Let the finite field GF(2^m) be defined by irreducible trinomial $f(x) = x^m + x^k + 1$, $\frac{m}{2} \leqslant k \leqslant m-1$. Then a bit-parallel Montgomery multiplier in GF(2^m) can be constructed from the expression (5), and one of the expressions (13), (21), (22) and (23). The complexity and time propagation delay are given as follows.*

1. *The complexity is m^2 AND gates and m^2-1 XOR gates. The incurred time delay is $T_A + (\lceil \log_2(m-2) \rceil + 2)T_X$, if $k = m-1$.*
2. *The complexity is m^2 AND gates and m^2-1 XOR gates. The incurred time delay is $T_A + (\lceil \log_2(m-\frac{k}{2}) \rceil + 2)T_X$, if $\frac{m+1}{2} \leqslant k \leqslant m-1$.*
3. *The complexity is m^2 AND gates and m^2-1 XOR gates. The incurred time delay is $T_A + (\lceil \log_2 k \rceil + 2)T_X$, if $k = \frac{m+1}{2}$.*
4. *The complexity is m^2 AND gates and $m^2 - \frac{m}{2}$ XOR gates. The incurred time delay is $T_A + (\lceil \log_2(m-1) + 1)T_X$, if $k = \frac{m}{2}$.*

4.7 Montgomery Squarer in GF(2^m)

When Algorithm 2 is used for squaring operation, only the first step needs to be changed. We rewrite Algorithm 2 for Montgomery squaring in GF(2^m) as follows

Algorithm 3. Generalized Montgomery squaring in GF(2^m)
Input: $a(x), r(x), f(x), f'(x)$
Output: $c(x) = a^2(x)r^{-1}(x) \bmod f(x)$
Step 1. $t(x) \Leftarrow a^2(x)$
Step 2. $u(x) \Leftarrow t(x)f'(x) \bmod r(x)$
Step 3. $\tilde{c}(x) \Leftarrow [t(x) + u(x)f(x)]/r(x)$
Step 4. If $\deg(\tilde{c}) > m-1$, then $c(x) \Leftarrow \tilde{c}(x) \bmod f(x)$, else $c(x) \Leftarrow \tilde{c}(x)$

With the same selection of the field $f(x) = x^m + x^k + 1$ and the fixed element $r(x) = x^k$, we proceed with Algorithm 3 step by step.

Step 1. From $t(x) = a^2(x)$, we have

$$\sum_{i=0}^{m-1} a_i x^{2i} = \sum_{i=0}^{2m-2} t_i x^i.$$

It can be seen from the above expression

$$t_i = \begin{cases} a_{\frac{i}{2}}, \; i = 0, 2, \ldots, 2m-2; \\ 0, \quad i = 1, 3, \ldots, 2m-3. \end{cases} \tag{24}$$

Not like multiplication, there is no bit operations needed here to obtain t_i.

Step 2-4. These three steps are very similar to these in Algorithm 2, and many intermediate results obtained in the last section can also be used here.
In the following we only consider the case that $k = m-1$ and m is even. For the other cases the deduction is similar. From (12) and (24), we have

$$\begin{array}{lll} (a) \quad t_L(x) = a_0 + a_1x^2 + a_2x^4 + \cdots + a_{\frac{m-2}{2}}x^{m-2} & [0, m-2] \\ (b) \; xt_L(x) = a_0x + a_1x^3 + \cdots + a_{\frac{m-2}{2}}x^{m-1} & [1, m-1] \\ (c) \quad t_H(x) = a_{\frac{m}{2}}x + a_{\frac{m+2}{2}}x^3 + \cdots + a_{m-1}x^{m-1} & [1, m-1] \end{array} \tag{25}$$

Note that the expression (a) in (25) has only even power terms and (b) and (c) have only odd power terms. Comparing (25) to (11) and note $c(x) = \tilde{c}(x)$ when $k = m - 1$, the coefficients c_i can be obtained as follows

$$c_i = \begin{cases} a_{\frac{i}{2}}, & i = 0, 2, \ldots, m-2; \\ a_{\frac{i-1}{2}} + a_{\frac{m+i-1}{2}}, & i = 1, 3, \ldots, m-1. \end{cases} \tag{26}$$

It can be seen that $\frac{m}{2}$ bit additions in GF(2) are required to compute c_i using (26). Then we know that to implement a bit-parallel Montgomery squarer needs only $\frac{m}{2}$ XOR gates. The time delay for this Montgomery squarer is equivalent to the delay of one XOR gate T_X.

The implementation results can be summarized as follows:

Theorem 2. *Let the finite field GF(2^m) be defined by irreducible trinomial $f(x) = x^m + x^k + 1$, $\frac{m}{2} \leqslant k \leqslant m-1$. Then a bit-parallel Montgomery squarer in GF(2^m) can be built with $\left\lceil \frac{m-1}{2} \right\rceil$ XOR gates and the time propagation delay is T_X.*

5 Comparison

Table 2. Comparison of Bit-Parallel Multipliers.

Proposals	# AND	# XOR	Time delay
$f(x) = x^m + x + 1$			
Wu, Hasan and Blake [6]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 m \rceil + 1)T_X$
Sunar and Koc [3]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 m \rceil + 1)T_X$
Wu [5]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 (m-2) \rceil + 2)T_X$
Presented here	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 (m-2) \rceil + 2)T_X$
$f(x) = x^m + x^k + 1,\ 1 < k < \frac{m}{2}$			
Wu, Hasan and Blake [6]	m^2	$m^2 - 1$	$T_A + \left(\left\lceil \log_2 \left\lceil \frac{m+k-1}{2} \right\rceil \right\rceil + 2 \right) T_X$
Sunar and Koc [3]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 m \rceil + 2)T_X$
Wu [5]	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 (m-1) \rceil + 2)T_X$
Presented here	m^2	$m^2 - 1$	$T_A + (\lceil \log_2 (m - \frac{k}{2}) \rceil + 2)T_X$
$f(x) = x^m + x^{\frac{m}{2}} + 1$			
Wu, Hasan and Blake [6]	m^2	$m^2 - \frac{m}{2}$	$T_A + \left\lceil \log_2 \left(m + 2 \left\lceil \frac{m}{4} \right\rceil \right) \right\rceil T_X$
Sunar and Koc [3]	m^2	$m^2 - \frac{m}{2}$	$T_A + (\lceil \log_2 m \rceil + 1)T_X$
Wu [5]	m^2	$m^2 - \frac{m}{2}$	$T_A + (\lceil \log_2 (m-1) \rceil + 1)T_X$
Presented here	m^2	$m^2 - \frac{m}{2}$	$T_A + (\lceil \log_2 (m-1) \rceil + 1)T_X$

Table 2 gives a comparison of four different implementations of bit-parallel multiplier in the same class of fields. Note that we consider the fields generated with two irreducible reciprocal trinomials are the same. The bit-parallel multiplier proposed by Wu, Hasan and Blake uses weakly dual basis (WDB) [6].[2] Sunar and Koc presented all trinomial Mastrovito multiplier using polynomial basis. The polynomial basis multiplier proposed in [5] has a different architecture from the Mastrovito multiplier.

It can be seen that all the multipliers achieve the same complexity in terms of the numbers of AND and XOR gates. The time propagation delay incurred with the multiplier presented here comparable to that of the previously proposed multipliers.

Table 3. Comparison of Polynomial Basis Bit-Parallel Squarers.

Proposals	# XOR	Time delay
$f(x) = x^m + x^k + 1$, where $m + k$ odd.		
Wu [5]	$\frac{m+k-1}{2}$	$2T_X$
Presented here	$\left\lceil \frac{m-1}{2} \right\rceil$	T_X
$f(x) = x^m + x^k + 1$, where both m and k are odd.		
Wu [5]	$\frac{m-1}{2}$	T_X
Presented here	$\frac{m-1}{2}$	T_X

It can be seen from Table 3 that Montgomery squarer has both lower complexity and lower time propagation delay for the case that $m+k$ is odd, compared to the regular polynomial basis squarer presented in [5].

References

1. C. K. Koc and T. Acar. Montgomery multiplication in GF(2^k). *Designs, Codes and Cryptography*, 14:57–69, 1998.
2. P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44:519–521, 1985.
3. B. Sunar and C. K. Koc. Mastrovito multiplier for all trinomials. *IEEE Trans. Comput.*, 48(5):522–527, 1999.
4. M. Wang and I. F. Blake. Bit serial multiplication in finite fields. *SIAM Discrete Mathematics*, 3(1):140–148, 1990.
5. H. Wu. Low-complexity arithmetic in finite field using polynomial basis. In *CHES'99*, pages 357–371. Springer-Verlag, 1999.
6. H. Wu, M. A. Hasan, and I. F. Blake. Low complexity weakly dual basis bit-parallel multiplier over finite fields. *IEEE Trans. Comput.*, 47(11):1223–1234, November 1998.

[2] A PB bit-parallel multiplier can be readily made by adding a basis conversion module to both the input and the output ends. By a theorem proposed in [4], when the field is generated with an irreducible trinomial, the coefficients of a field element in WDB is nothing but a permutation of the coefficients of the element in PB. Thus a weakly dual basis bit-parallel multiplier proposed in [4] can be used as a polynomial basis bit-parallel multiplier without additional gates and time delay.