On fully parallel Karatsuba Multipliers for $GF(2^m)$

F. Rodríguez-Henríquez
Computer Science Section, CINVESTAV-IPN
Av. IPN No. 2508, Col. San Pedro Zacatenco
México, D. F. 07360, México.
email: francisco@cs.cinvestav.mx

Ç. K. Koç
Department of Electrical & Computer Engineering,
Oregon State University, Corvallis, Oregon 97331
email: koc@ece.orst.edu

Abstract

In this paper we present a new approach that generalizes the classic Karatsuba multiplier technique. In contrast with versions of this algorithm previously discussed [1, 2], in our approach we do not use composite fields to perform the ground field arithmetic. The most attractive feature of the new algorithm presented here is that the degree of the defining irreducible polynomial can be arbitrarily selected by the designer, allowing the usage of prime degrees.

KEY WORDS
Karatsuba Multipliers, Galois Field $GF(2^m)$ Multipliers, Finite Field Arithmetic.

1 Introduction

Arithmetic over $GF(2^m)$ has many important applications, in particular in the theory of error control coding and in cryptography [1, 2, 3]. 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 circuit, whereas the time complexity is the total gate delay of the circuit.

Let $A(x), B(x)$ and $C'(x) \in GF(2^m)$ and $P(x)$ be the irreducible polynomial generating $GF(2^m)$. Multiplication in $GF(2^m)$ is defined as polynomial multiplication modulo the irreducible polynomial $P(x)$, $C'(x) = A(x)B(x) \bmod P(x)$. In order to obtain $C'(x)$, we can first obtain the product polynomial $C(x)$ of degree at most $2m-2$, as

$$C(x) = A(x)B(x) = (\sum_{i=0}^{m-1} a_i x^i)(\sum_{i=0}^{m-1} b_i x^i) \qquad (1)$$

In a second step the reduction operation needs to be performed in order to obtain the $m-1$ degree polynomial $C'(x)$, which is defined as

$$C'(x) = C(x) \pmod{P)(x}. \qquad (2)$$

Notice that once that the irreducible polynomial $P(x)$ has been selected, the reduction step can be accomplished by using XOR gates only.

Several architectures have been reported for multiplication in $GF(2^m)$. For example, efficient bit-parallel multipliers for both canonical and the normal basis representation have been proposed in [4, 5, 6]. All these algorithms exhibit a space complexity $O(m^2)$.

However, there are some asymptotically faster methods for finite field multiplications, such as the Karatsuba-Ofman algorithm [7, 8]. Discovered in 1962, it was the first algorithm to accomplish polynomial multiplication in under $O(m^2)$ operations [9]. Karatsuba multipliers can result in fewer bit operations at the expense of some design restrictions, particularly in the selection of the degree of the generating irreducible polynomial m.

In [7, 8, 5] was presented a Karatsuba multiplier based on composite fields of the type $GF((2^n)^s)$ with $m = sn$, $s = 2^t$, t an integer. However, for certain applications, especially, elliptic curve cryptosystems, it is important to consider finite fields $GF(2^m)$ where m is not necessarily a power of two. In fact, for this specific application some sources [10] suggest that, for security purposes, it is strongly recommended to choose degrees m for the finite field, in the range $[160, 512]$ with m a prime.

In this paper, we discuss some algorithms that implement generalized field Karatsuba multipliers in $GF(2^m)$, where m is an arbitrary integer. We present a modified version of the classic Karatsuba algorithm that we call binary Karatsuba multipliers. The organization of this paper is as follows: In § 2 we analyze the conventional Karatsuba polynomial multiplier technique for the particular case of $GF(2^m)$ with $m = 2^k n$, k an integer. In § 3 and § 4 we present a binary Karatsuba multiplier algorithm. Finally, in § 5 we discuss the implications of the results found in this research.

2 $2^k n$-bit Karatsuba Multipliers

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,

$$\begin{array}{rcl} A & = & \sum_{i=0}^{m-1} a_i x^i \ = \ \sum_{i=\frac{m}{2}}^{m-1} a_i x^i + \sum_{i=0}^{\frac{m}{2}-1} a_i x^i \\ & = & x^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} a_{i+\frac{m}{2}} x^i + \sum_{i=0}^{\frac{m}{2}-1} a_i x^i \ = \ x^{\frac{m}{2}} A^H + A^L \end{array}$$

and

$$\begin{array}{rcl} B & = & \sum_{i=0}^{m-1} b_i x^i \ = \ \sum_{i=\frac{m}{2}}^{m-1} b_i x^i + \sum_{i=0}^{\frac{m}{2}-1} b_i x^i \\ & = & x^{\frac{m}{2}} \sum_{i=0}^{\frac{m}{2}-1} b_{i+\frac{m}{2}} x^i + \sum_{i=0}^{\frac{m}{2}-1} b_i x^i \ = \ x^{\frac{m}{2}} B^H + B^L. \end{array}$$

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

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

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

$$\begin{array}{rcl} C & = & x^m A^H B^H + A^L B^L + \\ & & (A^H B^H + A^L B^L + (A^H + A^L)(B^L + B^H)) x^{\frac{m}{2}} \\ & = & x^m C^H + C^L. \end{array} \quad (4)$$

Let us define

$$\begin{array}{rcl} M_A & := & A^H + A^L; \\ M_B & := & B^L + B^H; \\ M & := & M_A M_B. \end{array} \quad (5)$$

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

$$\begin{array}{rcl} C^H & = & [c_{2m-2}, c_{2m-3}, \ldots, c_{m+1}, c_m]; \\ C^L & = & [c_{m-1}, c_{m-2}, \ldots, c_1, c_0]. \end{array} \quad (6)$$

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 elements $A, B \in GF(2^m)$ with $m = rn = 2^k n$, and where A, B can be expressed as,
$A = x^{\frac{m}{2}} A^H + A^L, B = x^{\frac{m}{2}} B^H + B^L$.
Output: A polynomial $C = AB$ with up to $2m-1$ coordinates, where $C = x^m C^H + C^L$.
Procedure Kmul2^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 $\frac{r}{2} - 1$ do
5.            $M_{A\,i} = A_i^L + A_i^H$;
6.            $M_{B\,i} = B_i^L + B_i^H$;
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.           $C_{\frac{r}{2}+i} \ = \ C_{\frac{r}{2}+i} + M_i$;
16.     end
17. end
```

Figure 1. $m = 2^k n$-bit Karatsuba multiplier.

Eventually, after no more than $\lceil \log_2(m) \rceil$ iterations, all the polynomial operands collapse into single coefficients. In the last iteration, the resulting bit multiplications can be directly computed. Although it is possible to implement the Karatsuba algorithm until the $\lceil \log_2 m \rceil$ iteration, it is usually more practical to truncate the algorithm earlier. If the Karatsuba algorithm is truncated at a certain point, the remaining multiplications can be computed by using alternative techniques (classic algorithm, Mastrovito multipliers, and other techniques). The best results are then obtained by using hybrid techniques, i.e., we use Karatsuba to reduce the multiplier complexity of relatively big size operands, followed by efficient algorithms to compute the multipliers for small size operands.

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

$$\begin{array}{rcl} \#\text{XORs} & \leq & \left(\frac{m}{n}\right)^{\log_2 3} (8\frac{m}{r} - 2 + M_{xor2^n}) \\ & & -8m + 2; \\ \#\text{ANDs} & \leq & \left(\frac{m}{n}\right)^{\log_2 3} M_{and2^n}; \\ \text{Delay} & \leq & T_{delay2^n} + 4T_X \log_2\left(\frac{m}{n}\right). \end{array} \quad (7)$$

In this case it has been assumed that the block selected to implement the $GF(2^n)$ arithmetic has a T_{delay2^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 n-bit multiplier are given as

$$\begin{aligned} \text{\# XORs} &= (n-1)^2 \;; \\ \text{\# ANDs} &= n^2 \;; \\ \text{Delay} &\leq T_{AND} + T_X \lceil \log_2 n \rceil \;. \end{aligned} \tag{8}$$

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-bit multiplicand level are upper bounded by

$$\begin{aligned} &\text{\# XORs} \leq \left(\frac{m}{n}\right)^{\log_2 3} (n^2 + 6n - 1) - 8m + 2 \;; \\ &\text{\# ANDs} \leq 3^{\log_2 r} M_{and2^n} \;=\; \left(\frac{m}{n}\right)^{\log_2 3} n^2; \\ &\quad \text{Delay} \leq T_{AND} + T_X(\log_2 n + 4\log_2 r) \;. \end{aligned} \tag{9}$$

Table 1 shows the space and time complexities for the hybrid Karatsuba multiplier for the optimal case where m is a power of two. The values of m presented in table 1 are the first ten powers of two. Notice that the multipliers for $m = 1, 2, 4$ are assumed to be implemented using the classic method only. As we will see in § 3, the complexities of the hybrid Karatusba multiplier for degrees $m = 2^k$ happen to be crucial to find the hybrid Karatsuba complexities for arbitrary degrees of m.

m	r	n	AND gates	XOR gates	Time delay
1	1	1	1	0	T_A
2	1	2	4	1	$T_X + T_A$
4	1	4	16	9	$2T_X + T_A$
8	2	4	48	55	$6T_X + T_A$
16	4	4	144	225	$10T_X + T_A$
32	8	4	432	799	$14T_X + T_A$
64	16	4	1296	2649	$18T_X + T_A$
128	32	4	3888	8455	$22T_X + T_A$
256	64	4	11664	26385	$26T_X + T_A$
512	128	4	34992	81199	$30T_X + T_A$

Table 1. Space and time complexities for several $m = 2^k$-bit hybrid Karatsuba multipliers.

3 Binary Karatsuba Multipliers

In the last section we analyzed hybrid Karatsuba multipliers for composite degrees of m. However, as it was mentioned in §1, for cryptographic applications it is advisable to use finite fields of prime degree. Therefore, it is important to investigate a generalized version of the algorithm in figure 1 that is not restricted to special composite degrees of m. In this section we present an algorithm that allows us to implement a binary Karatsuba multiplier for arbitrary degrees of m. We also give a rigorous analysis of its corresponding space and time complexities.

3.1 Binary Karatsuba Strategy

Let us consider the multiplication of two polynomials $A, B \in GF(2^m)$, such that their degree is less or equal to $m-1$, where $m = 2^k + d$. As a very first approach, we could pretend that both operands have 2^{k+1} coordinates each, where their respective $2^{k+1} - d$ most significant bits are all equal to zero. If we partition the operands A and B into two halfes, then, in order to compute their polynomial multiplication, we can use the algorithm in figure 1 with $m = 2^{k+1}$. Although this approach is a valid one, it clearly implies the waste of several arithmetic operations, as some of the most significant bits of the operands are zeroes. However, if we were able to identify the extra arithmetic operations and remove them from the algorithm in figure 1, we would then be able to find a quasi-optimal solution for arbitrary degrees of m. To see how this can be done, consider the algorithm shown in figure 2, which has been adapted from the one presented in figure 1.

Input: Two elements $A, B \in GF(2^m)$ with m an arbitrary number, and where A, B can be expressed as $A = x^{\frac{m}{2}}A^H + A^L, B = x^{\frac{m}{2}}B^H + B^L$.
Output: A polynomial $C = AB$ with up to $2m-1$ coordinates, where $C = x^m C^H + C^L$.
Procedure mulgen_m(C, A, B)

```
0. begin
1.      $k = \lfloor \log_2 m \rfloor$;
2.      $d = m - 2^k$;
3.      if ($d == 0$) then
4.         $C = Kmul2^k(A, B)$;
5.         return;
6.         for i from 0 to $d-1$ do
7.         $M_{A_i} = A_i^L + A_i^H$;
8.         $M_{B_i} = B_i^L + B_i^H$;
9.         end
10.     $mul2^k(C^L, A^L, B^L)$;
11.     $mul2^k(M, M_A, M_B)$;
12.     $mulgen\_d(C^H, A^H, B^H)$;
13.     for i from 0 to $2k-2$ do
14.           $M_i = M_i + C_i^L + C_i^H$;
15.     end
16.     for i from 0 to $2k-2$ do
17.           $C_{k+i} = C_{k+i} + M_i$;
18.     end
19. end
```

Figure 2. m-bit binary Karatsuba multiplier.

In lines 1-2 the values of the constants k, d such that $m = 2^k + d$ are computed. If $d = 0$, i.e, if m is a power of two, then the binary Karatsuba algorithm of figure 2 reverts to the specialized algorithm in figure 1 presented in the previous section. If that is not the case, our algorithm uses the constants k and d to prevent us to compute unnecessary arithmetic operations. In lines 6-9, the d least significant bits of M_A and M_B of equation (5) are computed using the d non-zero coordinates of A^H and B^H. The remaining $k-d$ most significant bits of M_A and M_B are directly obtained from A^L and B^L, respectively. Notice that the operands, A^L, B^L, M_A and M_B are all 2^k-bit polynomials. Because of that, our algorithm invokes the multiplier of figure 1 in lines 10 and 11. On the other hand, both operands A^H and B^H are d-bit polynomials, where d, in general, is not a power of two. Consequently, in line 12, the algorithm calls itself in a recursive manner. This recursive call is invoked using the operand's degree reduced to d. Clearly in each iteration the degree of the operands gets reduced, and eventually, after a total of h iterations (where h is the hamming weight of the binary representation of the original degree m), the algorithm ends.

It can be shown that the space complexity of the m-bit binary Karatsuba multiplier of figure 2 is given as,

$$\begin{aligned}\text{XOR Gates} &= \sum_{i=0}^{h-2}[2MUL_X(2^{k_i})+ \\ &\quad \min\left(4(k_i+d_{i+1}-1),\ k_i+10d_{i+1}-4\right)] \\ &\quad + MUL_X(2^{k_{h-1}}) \\ \text{AND Gates} &= \sum_{i=0}^{h-2}\left(2MUL_A(2^{k_i})\right)+MUL_X(2^{k_{h-1}}).\end{aligned} \tag{10}$$

Where

$$\begin{aligned} d_0 &= m; \\ k_0 &= \lfloor \log_2 m \rfloor; \\ d_i &= d_{i-1} - 2^{k_{i-1}}; \\ k_i &= \lfloor \log_2 d_i \rfloor; \\ h &= hamming\_weight(m). \end{aligned} \tag{11}$$

$MUL_X(r)$ and $MUL_A(r)$ represent the XOR gate and the AND gate complexities of the algorithm in figure 1, respectively, with $r = 2^k$, k an integer.

Whereas, the associated time delay of the m-bit binary Karatsuba multiplier of figure 2, with m not a power of two, is given as

$$\text{Time Delay} = MUL_{delay}(2^{\lfloor \log_2 m \rfloor}) + 4T_X; \tag{12}$$

Where $MUL_{delay}(r)$ represents the time complexity of the algorithm in figure 1, with $r = 2^k$, k an integer.

4 Binary Karatsuba Multipliers Revisited

The algorithm presented in figure 2 achieves the goal of obtaining a modular version of the Karatsuba algorithm for arbitrary degrees of m. However, that algorithm is inefficient for certain degrees of m, especially for the cases when $d << 2^k$, $m = 2^k + d$. Fortunately, there is an alternative approach that can help us alleviate this problem. To see this, let us examine again the algorithm in figure 2. In line 11 the algorithm obtains the value of the polynomial M of equation (5) via polynomial multiplication using a 2^k-bit multiplier block. If $k > 2$, the space cost of a 2^k-bit multiplier block is upper bounded by

$$\begin{aligned} \text{XOR Gates} &\leq 13\cdot 3^{k-1} - 2^{k+3} + 2; \\ \text{AND Gates} &\leq 16\cdot 3^{k-2}. \end{aligned} \tag{13}$$

However, if $d << 2^k$, there is a more efficient way to obtain M. From equation (5), we can reformulate M as,

$$\begin{aligned} M = M_A M_B = (A^H + A^L)(B^L + B^H) = \\ = A^L B^L + A^L B^H + A^H B^L + A^H B^H. \end{aligned} \tag{14}$$

By applying the classic multiplier algorithm mentioned previously, we can compute the products $A^L B^H$ and $A^H B^L$ at the space cost of $(2^k - 1)(d-1)$ XOR gates and $2^k d$ AND gates each. Thus, using equation (14), the polynomial M can be obtained at a space cost given as

$$\begin{aligned} \text{XOR Gates} &= 2(2^k-1)(d-1) + 2(2^k+d-1)+ \\ &\quad + (2d-1) \\ &= 2d(2^k+1) - 1; \\ \text{AND Gates} &= 2d2^k. \end{aligned} \tag{15}$$

If we compute M using this approach, we do not need to compute the loop in lines 6-9 of the algorithm in figure 2 anymore, yielding an extra saving of d XOR gates. Considering this, and by comparing equations (13) and (15), we conclude that the polynomial M can be computed more efficiently by using the equation (14) rather than using a 2^k-bit Karatsuba multiplier, if the following condition is satisfied

$$d < \left\lfloor \frac{13\cdot 3^{k-1} - 2^{k+3} + 3}{2^{k+1}+1} \right\rfloor \tag{16}$$

where $m = 2^k + d$, $k > 2$.

Figure 3 shows the algorithm for the binary Karatsuba multiplier if condition (16) is satisfied. Notice that equation (14) is implemented in line 8 of the algorithm. In practice, for a given arbitrary degree m, the designer has the option to compute the polynomial M either implementing equation (14) or using a 2^k-bit Karatsuba multiplier block. The best option can be selected after evaluating condition (16). In the next section we present a design example that illustrates this design process.

4.1 An Example

As a design example, let us consider the polynomial multiplication of the elements A and $B \in GF(2^{193})$. Since $(193)_2 = 11000001$, the Hamming weight h of the binary representation of m is $h = 3$. This implies that we need a total of three iterations in order to compute the multiplication using the generalized m-bit binary Karatsuba multiplier. Additionally, we notice that for this case $m = 193 = 2^7 + 65$.

Input: Two elements $A, B \in GF(2^m)$ with $m = 2^k + d$, $k > 2$, and where A, B can be expressed as
$A = x^{\frac{m}{2}} A^H + A^L, B = x^{\frac{m}{2}} B^H + B^L$.
Output: A polynomial $C = AB$ with up to $2m - 1$ coordinates, where $C = x^m C^H + C^L$.

```
Procedure mulgen_m(C, A, B)
0. begin
1.     $k = \lfloor \log_2 m \rfloor$;
2.     $d = m - 2^k$;
3.     if ($d == 0$) then
4.         $C = Kmul2^k(A, B)$;
5.         return;
6.     $mul2^k(C^L, A^L, B^L)$;
7.     $mulgen\_d(C^H, A^H, B^H)$;
8.     $M = C^L + mul\_classic(A^L B^H) +$
              $+ mul\_classic(A^H B^L) + C^H$;
9.     for i from 0 to $2k - 2$ do
10.        $M_i = M_i + C_i^L + C_i^H$;
11.    end
12.    for i from 0 to $2k - 2$ do
13.        $C_{k+i} = C_{k+i} + M_i$;
14.    end
15. end
```

Figure 3. m-bit binary Karatsuba multiplier if condition (16) holds.

Using equation (11), we find $k_0 = 7$ and $d_1 = 65$. Therefore, condition (16) yields

$$d_1 = 65 > \left\lfloor \frac{13 \cdot 3^{k_0 - 1} - 2^{k_0 + 3} + 3}{2^{k_0+1} + 1} \right\rfloor = 32.$$

Thus, in the first iteration it is preferable to obtain M using a 128-bit Karatsuba multiplier block. In this first iteration of the algorithm, we need a total of $2d_1 = 130$ and $2(2^{k_0+1} + d_1) - 4 = 638$ XOR gates, in order to implement the first loop (lines 9-12) and the second loop (lines 16-19) of figure 2, respectively. We also need to use two $2^{k_0} = 128$-bit multiplier blocks, as well as one mulgen$R(65)$ multiplier block. The latter multiplier is implemented in the second iteration. In the second iteration we have $c_1 = 65 = 2^6 + 1$, yielding $k_1 = 6$ and $d_2 = 1$. From condition (16) we see that $d_2 = 1 < \left\lfloor \frac{13 \cdot 3^{k_1 - 1} - 2^{k_1+3} + 3}{2^{k_1+1}+1} \right\rfloor = 20$. Thus, in the second iteration it is better to compute M using equation (14). Therefore, in the second and third iterations we need one 64-bit Karatsuba multiplier block, 2 classical multiplier blocks, 68 XOR gates and 1-bit multiplier.

We can compute the space and time complexities of the generalized $m = 193$-bit binary Karatsuba multiplier using equations (10), (12) and (15). We also use the complexity figures listed in table 1, obtaining

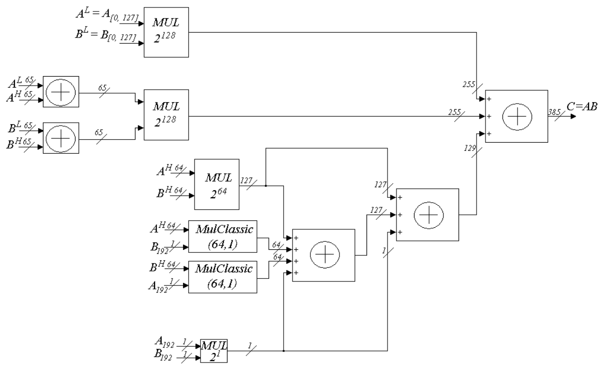

Figure 4. Schematic diagram of a generalized $m = 193$-bit binary Karatsuba multiplier

$$\begin{aligned}
\text{\# XORs} &= 768 + 2MUL_X(128) + 68 + MUL_X(64) \\
&\quad + 2mul\_classic_X(64,1) + MUL_X(1) \\
&= 768 + 2 \cdot 8455 + 68 + 2649 + 129 + 0 \\
&= 20524. \\
\text{\# ANDs} &= 2MUL_A(128) + MUL_A(64) + \\
&\quad + 2mul\_classic_X(64,1) + MUL_A(1) \\
&= 2 \cdot 3888 + 1296 + 128 + 1 = 9201. \\
\text{Delay} &= MUL_{delay}(2^{\lfloor \log_2 m \rfloor}) + 4T_X \\
&= MUL_{delay}(2^{\lfloor \log_2 193 \rfloor}) + 4T_X \\
&= 26T_X + T_A.
\end{aligned}$$

For most hardware implementation schemes of digital logic, it is reasonable to assume that the area costs of an AND gate and an XOR gate are of about 1.26 units and 2.2 units, respectively, where an area of 1 represents the area cost of a NAND gate. Based on this assumption, we show in figure 5 the estimated total area of the binary Karatsuba multiplier in the range mentioned above, together with the corresponding area estimation for the hybrid Karatsuba multiplier (using $n = 1$), that was presented in § 2.

5 Conclusions and Discussion of the Results

In this paper we presented a new approach that generalizes the classic Karatusba multiplier technique. The most attractive features of the new algorithm presented here is that the degree of the defining irreducible polynomial can be arbitrarily selected by the designer, allowing the usage of prime degrees. In addition, the new field multiplier leads to architectures which show a considerably improved gate complexity when compared to traditional approaches. Finally, the new multiplier leads to highly modular architectures and is thus well suited for VLSI implementations.

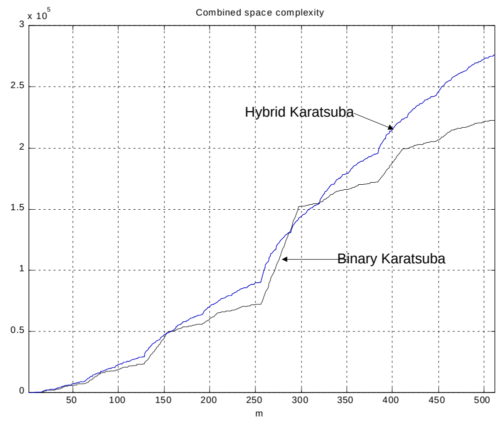

Figure 5. Total area complexity of the modified binary and hybrid Karatsuba multipliers

References

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

[2] A. J. Menezes, I. F. Blake, X. Gao, R. C. Mullen, S. A. Vanstone, and T. Yaghoobian. *Applications of Finite Fields*. Kluwer Academic Publishers, Boston, MA, 1993.

[3] S. B. Wicker and V. K. Bhargava, editors. *Reed-Solomon Codes and Their Applications*. Prentice-Hall, Englewood Cliffs, NJ, 1994.

[4] M. A. Hasan, M. Z. Wang, and V. K. Bhargava. A modified Massey-Omura parallel multiplier for a class of finite fields. *IEEE Transactions on Computers*, 42(10):1278–1280, November 1993.

[5] C. Paar. *Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields*. PhD thesis, Universität GH Essen, VDI Verlag, 1994.

[6] B. Sunar and Ç. K. Koç. Mastrovito multiplier for all trinomials. *IEEE Transactions on Computers*, 48(5):522–527, May 1999.

[7] C. Paar. A new architecture for a paralel finite field multiplier with low complexity based on composite fields. *IEEE Transactions on Computers*, 45(7):856–861, July 1996.

[8] J. Guajardo and C. Paar. Efficient algorithms for elliptic curve cryptosystems. In B. S. Kaliski Jr., editor, *Advances in Cryptology — CRYPTO 97*, Lecture Notes in Computer Science, No. 1294, pages 342–356. Springer-Verlag, Berlin, Germany, 1997.

[9] K. Geddes, S. Czapor, and G. Labahn. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Boston, MA, 1992.

[10] IEEE P1363. Standard specifications for public-key cryptography. Draft Version 7, September 1998.