Fast arithmetic for polynomials over $\mathbb{F}_2$ in hardware

Joachim von zur Gathen
B-IT - Universität Bonn
Email: gathen@bit.uni-bonn.de

Jamshid Shokrollahi
B-IT - Universität Bonn
Email: jamshid@bit.uni-bonn.de

***Abstract*— We study different possibilities of implementing Karatsuba multipliers for polynomials over $\mathbb{F}_2$ on Field Programmable Gate Arrays (FPGAs).**

This is a core task for implementing finite fields of characteristic 2. Algorithmic and platform dependent optimizations yield efficient hardware designs. The resulting structure is hybrid in two different aspects. On the one hand, a combination of various methods decreases the number of bit operations. On the other hand, a mixture of sequential and combinational circuit design techniques including pipelining is used to design a circuit which can be adapted flexibly to time-area constraints. The approach—both theory and implementation—can be viewed as a further step towards taming the machinery of fast algorithmics for hardware applications.

I. Introduction

Arithmetic in finite fields is a central algorithmic task in cryptography. There are two types of groups associated to such fields: their multiplicative group of invertible elements, and elliptic (or hyperelliptic) curves. These can then be used in group-based cryptography, relying on the difficulty of computing discrete logarithms. Here we focus on fields of characteristic 2. The most fundamental task in arithmetic is multiplication. In our case, this amounts to multiplication of polynomials over $\mathbb{F}_2$, followed by a reduction modulo the fixed polynomial defining the field extension. This reduction can itself be performed by using multiplication routines or by a small hardware circuit when the polynomial is sparse. A trinomial can be used in many cases, and it is conjectured that otherwise a pentanomial can be found (see von zur Gathen & Nöcker (2006)). As to the other arithmetic operations, addition is bitwise XORing of vectors, squaring a special case of multiplication (much simplified by using a normal basis), and inversion more expensive and usually kept to a minimum.

Classical methods to multiply two n-bit polynomials require $O(n^2)$ bit operations. The Karatsuba algorithm reduces this to $O(n^{\log_2 3})$, and fast Fourier transformations to $O(n \log n \operatorname{loglog} n)$. The Cantor multiplier with a cost of $O(n(\log n)^2(\operatorname{loglog} n)^3)$ is designed for fields of characteristic 2, but we do not study it here (see Cantor (1989) and von zur Gathen & Gerhard (1996)). Traditional lore held that asymptotically fast methods are not suitable for hardware. We disprove this view in the present paper, continuing the work by Grabbe et al. (2003).

Our methods are asymptotically good and thus efficient for large degrees. Sophisticated implementation strategies decrease the crossover points between different algorithms and make them efficient for practical applications. Much care is required for software implementations (see von zur Gathen & Gerhard (2003), chapter 8, and Shoup's NTL software). The Karatsuba method has the lowest crossover point with the classical algorithm.

The Karatsuba algorithm for multiplication of large integers was introduced by Karatsuba & Ofman (1963). It is commonly used recursively, but here we combine it with other algorithms for hardware realization of cryptographic tasks.

FPGAs provide useful implementation platforms for cryptographic algorithms both for prototyping where early error finding is possible, and as systems on chips where system parameters can easily be changed to satisfy evolving security requirements.

Efficient software implementations of Karatsuba multipliers using general purpose processors have been discussed thoroughly in the literature (see Paar (1994), Bailey & Paar (1998), Koç and Erdem (2002), Hankerson et al. (2003), chapter 2, and von zur Gathen & Gerhard (2003), chapter 8), but hardware implementations have attracted less attention. The only works known to us are Jung et al. (2002), Weimerskirch & Paar (2003), and Grabbe et al. (2003). Jung et al. (2002) and Weimerskirch & Paar (2003) suggest to use algorithms with $O(n^2)$ operations to multiply polynomials which contain a prime number of bits. Their proposed number of bit operations is by a constant factor smaller than the classical method but asymptotically larger than that for the Karatsuba method. Grabbe et al. (2003) propose a hybrid implementation of the Karatsuba method which reduces the latency by pipelining and by mixing sequential and combinational circuits.

The present work tries to decrease the resource usage of polynomial multipliers using both known algorithmic and platform dependent methods. Our Table II presents the best choice of hybrid multiplication algorithms for polynomials with at most 8192 bits, as long as the choice is restricted to six (recursive) methods, namely classical, Karatsuba, a variant of Karatsuba for quadratic polynomials, and three other methods proposed by Montgomery (2005). The "best" refers to minimizing the area measure. This is an algorithmic and machine independent optimization. In an earlier implementation (Grabbe et al. (2003)) we had designed a 240-bit multiplier on a XC2V6000-4FF1517-4 FPGA. We re-use this structure and decrease its area and time by using the methods developed in Section III and better usage of pipelining deployed in this structure. Using this 240-bit multiplier we cover in particular the 233-bit polynomials proposed by NIST for elliptic curve cryptography in the Digital Signature Standard (DSS) (2000).

The structure of this paper is as follows. First the Karatsuba method and its cost are studied in Section II. Section III is devoted to optimized hybrid Karatsuba implementations. Section IV shows how a hybrid structure and pipelining together with the reduction of number of recursion levels improves resource usage in the circuit from Grabbe et al. (2003) and Section V concludes the paper.

Parts of this paper have appeared in von zur Gathen & Shokrollahi (2005). The inclusion of Montgomery multiplication in Table II and the corresponding considerations are presented here for the first time.

II. The Karatsuba algorithm

The three coefficients of the product $(a_1x+a_0)(b_1x+b_0) = a_1b_1x^2 + (a_1b_0 + a_0b_1)x + a_0b_0$ are "classically" computed with 4 multiplications and 1 addition from the four input coefficients a_1, a_0, b_1, and b_0. The following formula uses only 3 multiplications and 4 additions:

$$(a_1x + a_0)(b_1x + b_0) = a_1b_1x^2 + ((a_1 + a_0)(b_1 + b_0) - a_1b_1 - a_0b_0)x + a_0b_0. \tag{1}$$

We call this the 2-*segment Karatsuba method or* K_2. Setting $m = \lceil n/2 \rceil$, two n-bit polynomials (thus of degrees less than n) can be rewritten and multiplied using the formula:

$$(f_1x^m + f_0)(g_1x^m + g_0) = h_2x^{2m} + h_1x^m + h_0,$$

where f_0, f_1, g_0, and g_1 are m-bit polynomials respectively. The polynomials h_0, h_1, and h_2 are computed by applying the Karatsuba algorithm to the polynomials f_0, f_1, g_0, and g_1 as single coefficients and adding coefficients of common powers of x together. This method can be applied recursively. The circuit to perform a single stage is shown in Figure 1.

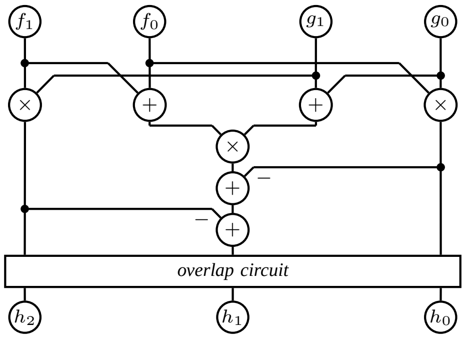
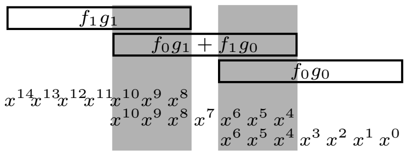


Fig. 1. One level of the Karatsuba multiplication

The "overlap circuit" adds common powers of x in the three generated products. For example if $n = 8$, then the input polynomials have degree at most 7, each of the polynomials f_0, f_1, g_0, and g_1 is 4 bits long and thus of degree at most 3, and their products will be of degree at most 6. The effect of the overlap module in this case is represented in Figure 2, where coefficients to be added together are shown in the same columns.

Figures 1 and 2 show that we need three multiplication calls at size $m = \lceil n/2 \rceil$ and some adders: 2 for input, 2 for output, and 2 for the overlap module of lengths m, $2m - 1$, and $m - 1$ respectively. Below we consider various algorithms A of a similar structure. We denote the size reduction factor, the number of multiplications, input adders, output adders, and the total number of bit operations to multiply two n-bit polynomials in A by b_A, mul_A, ia_A, oa_A, and $\mathsf{M}_A(n)$, respectively. Then



f_1g_1
$f_0g_1 + f_1g_0$
f_0g_0
$x^{14} x^{13} x^{12} x^{11} x^{10} x^9 \; x^8$
$x^{10} x^9 \; x^8 \; x^7 \; x^6 \; x^5 \; x^4$
$x^6 \; x^5 \; x^4 \; x^3 \; x^2 \; x^1 \; x^0$

Fig. 2. The overlap circuit for the 8-bit Karatsuba multiplier

$$\mathsf{M}_A(n) = \mathsf{mul}_A \; \mathsf{M}(m) + \mathsf{ia}_A \; m + \mathsf{oa}_A \; (2m-1) + 2(\mathsf{b}_A - 1)(m-1), \tag{2}$$

where $m = \lceil n/\mathsf{b}_A \rceil$ and $\mathsf{M}(m)$ is the cost of the multiplication call for m-bit polynomials. For $A = K_2$, this becomes:

$$\mathsf{M}_{K_2}(n) = 3 \; \mathsf{M}(m) + 8m - 4, \quad m = \lceil n/2 \rceil.$$

Our interest is not the usual recursive deployment of this kind of algorithms, but rather the efficient interaction of various methods. We include in our study the classical multiplication C_b on b-bit polynomials and algorithms for 3, 5, 6, and 7-segment polynomials which we call K_3 (3-segment Karatsuba, see Blahut (1985), Section 3.4, page 85), M_5, M_6, and M_7 (see Montgomery (2005)). The parameters of these algorithms are given in Table I.

TABLE I

The parameters of some multiplication methods

Algorithm A	b_A	mul_A	ia_A	oa_A
K_2	2	3	2	2
K_3	3	6	6	6
M_5	5	13	22	30
M_6	6	17	61	40
M_7	7	22	21	55
$C_b, b \geq 2$	b	b^2	0	$(b-1)^2$

III. Hybrid design

For fast multiplication software, a judicious mixture of table look-up and classical, Karatsuba and even faster (FFT) algorithms must be used (see von zur Gathen & Gerhard (2003), chapter 8, and Hankerson et al. (2003), chapter 2). Suitable techniques for hardware implementations are not thoroughly studied in the literature. In contrast to software implementations where the word-length of the processor, the datapath, and the set of commands are fixed, hardware designers have more flexibility. In software solutions speed and memory usage are the measures of comparison whereas hardware implementations are generally designed to minimize the area and time, simultaneously or with some weight-factors. In this section we determine the least-cost combination of any basic routines for bit sizes up to 8192. Here, cost corresponds to the total number of operations in software, and the area in hardware. Using pipelining and the structure of Grabbe et al. (2003)

this can also result in multipliers which have small time-area parameters.

We present a general methodology for this purpose. We start with a toolbox $\mathcal{T}$ of basic algorithms, namely $\mathcal{T} = \{\text{classical}, K_2, K_3, M_5, M_6, M_7\}$. Each $A \in \mathcal{T}$ is defined for b_A-bit polynomials. We denote by $\mathcal{T}^*$ the set of all iterated (or hybrid algorithms) compositions from $\mathcal{T}$; this includes $\mathcal{T}$ and the identity. Figure 3 shows the hierarchy of a hybrid algorithm for 12-bit polynomials using our toolbox $\mathcal{T}$. At the top level, K_2 is used, meaning that the 12-bit input polynomials are divided into two 6-bit polynomials each and K_2 is used to multiply the input polynomials as if each 6-bit polynomial were a single coefficient. K_2C_3 performs the three 6-bit multiplications. One of these 6-bit multipliers is circled in Figure 3 and unravels as follows:

$$\begin{aligned}
&(a_5x^5 + \cdots + a_0) \cdot (b_5x^5 + \cdots + b_0) = \\
&((a_5x^2 + a_4x + a_3)x^3 + (a_2x^2 + a_1x + a_0)) \\
&\cdot ((b_5x^2 + b_4x + b_3)x^3 + (b_2x^2 + b_1x + b_0)) = \\
&(A_1x^3 + A_0) \cdot (B_1x^3 + B_0) = A_1B_1x^6 + \\
&((A_1 + A_0)(B_1 + B_0) - A_1B_1 - A_0B_0)x^3 + A_0B_0
\end{aligned}$$

Each of A_1B_1, $(A_1 + A_0)(B_1 + B_0)$, and A_0B_0 denotes a multiplication of 3-bit polynomials and will be done classically using the formula

$$\begin{aligned}
&(a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0) = a_2b_2x^4 + \\
&(a_2b_1 + a_1b_2)x^3 + (a_2b_0 + a_1b_1 + a_0b_2)x^2 + \\
&(a_1b_0 + a_0b_1)x + a_0b_0.
\end{aligned}$$

Thick lines under each C_3 indicate the nine 1-bit multiplications to perform C_3. We designate this algorithm, for 12-bit polynomials, with $K_2K_2C_3 = K_2^2C_3$ where the left hand algorithm, in this case K_2, is the topmost algorithm.

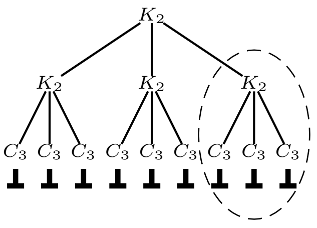


Fig. 3. The multiplication hierarchy for $K_2K_2C_3$

As in (2), the cost of a hybrid algorithm $A_2A_1 \in \mathcal{T}^*$ with $A_1, A_2 \in \mathcal{T}^*$ satisfies

$$\begin{aligned}
\mathsf{M}_{A_2A_1}(n) \leq &\mathsf{mul}_{A_2}\ \mathsf{M}_{A_1}(m) + \mathsf{ia}_{A_2}\ m + \\
&\mathsf{oa}_{A_2}\ (2m-1) + 2(\mathsf{b}_{A_2} - 1)(m-1), \qquad (3)
\end{aligned}$$

where $\mathsf{M}_A(1) = 1$ for any $A \in \mathcal{T}^*$ and $m = \lceil n/(\mathsf{b}_{A_2}\mathsf{b}_{A_1})\rceil = \lceil\lceil n/\mathsf{b}_{A_2}\rceil/\mathsf{b}_{A_1}\rceil$. Each $A \in \mathcal{T}^*$ has a well-defined input length b_A, given in Table I for basic tools and by multiplication for composite methods. We extend the notion by applying A also to fewer than b_A bits, by padding with leading zeros, so that $\mathsf{M}_A(m) = \mathsf{M}_A(\mathsf{b}_A)$ for $1 \leq m \leq \mathsf{b}_A$. For some purposes, one might consider the savings due to such a-priori-zero coefficients. Our goal, however, is a pipelined structure where such a consideration cannot be incorporated. The minimum hybrid cost over $\mathcal{T}$ is

$$\mathsf{M}(n) = \min_{A \in \mathcal{T}^*, \mathsf{b}_A \geq n} \mathsf{M}_A(n).$$

We first show that the infinitely many classical algorithms in $\mathcal{T}$ do not contribute to optimal methods beyond size 12.

Lemma 1: For $A \in \mathcal{T}^*$ and integers $m \geq 1$ and $b, c \geq 2$ we have the following.

(*i*) $\mathsf{M}_{C_bC_c}(bc) = \mathsf{M}_{C_{bc}}(bc)$.
(*ii*) $\mathsf{M}_{C_bA}(\mathsf{b}_Abm) \geq \mathsf{M}_{AC_b}(\mathsf{b}_Abm)$.
(*iii*) *For any n, there is an optimal hybrid algorithm all of whose components are non-classical, except possibly the right most one.*
(*iv*) *If $n \geq 13$, then C_n is not optimal.*

We now present a dynamic programming algorithm which computes an optimal hybrid algorithm from $\mathcal{T}^*$ for n-bit multiplication, for $n = 1, 2, \ldots$.

Algorithm 1 Finding optimal algorithms in $\mathcal{T}^*$

```
Input: The toolbox $\mathcal{T} = \{\text{classical}, K_2, K_3, M_5, M_6, M_7\}$
    and an integer $N$.
Output: Table $T$ with $N$ rows containing the optimal algo-
    rithms for $1 \leq n \leq N$ and their costs.
 1: Enter the classical algorithm and cost 1 for $n = 1$ into $T$
 2: for $n = 2, \ldots, N$ do
 3:     $bestalgorithm \leftarrow$ unknown, $mincost \leftarrow$ +infinity
 4:     for $A = K_2, \ldots, M_7$ do
 5:         Compute $\mathsf{M}_A(n)$ according to (2)
 6:         if $\mathsf{M}_A(n) < mincost$ then
 7:             $bestalgorithm \leftarrow A$, $mincost \leftarrow \mathsf{M}_A(n)$
 8:         end if
 9:     end for
10:     if $n < 13$ then
11:         $\mathsf{M}_{C_n} \leftarrow 2n^2 - 2n + 1$
12:         if $\mathsf{M}_{C_n}(n) < mincost$ then
13:             $bestalgorithm \leftarrow C_n$, $mincost \leftarrow \mathsf{M}_{C_n}(n)$
14:         end if
15:     end if
16:     Enter $bestalgorithm$ and $mincost$ for $n$ into $T$
17: end for
```

Theorem 2: Algorithm 1 works correctly as specified. The operations (arithmetic, table look-up) have integers with $O(\log N)$ bits as input, and their total number is $O(N)$.

The optimal recursive method for each polynomial length up to 8192 is shown in Table II. The column "length" of this table represents the length (or the range of lengths) of polynomials for which the method specified in the column "method" must be used. As an example, for 194-bit polynomials the method M_7 is used at the top level. This requires 22 multiplications of polynomials with $\lceil 194/7 \rceil = 28$ bits, which are done by means of K_2 on top of 14-bit polynomials. These 14-bit multiplications are executed again using K_2 and finally polynomials of length 7 are multiplied classically. Thus the

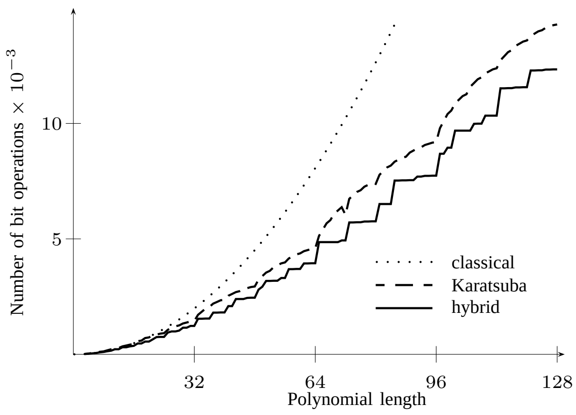


Fig. 4. The number of bit operations of the classical, recursive Karatsuba, and the hybrid methods to multiply polynomials of degree smaller than 128

TABLE II

OPTIMAL MULTIPLICATIONS FOR POLYNOMIAL LENGTHS UP TO 8192

length	method	length	method	length	method
$1-5$	$C_1 - C_5$	$301-320$	K_2	$1603-1610$	M_5
6	K_2	$321-343$	M_7	$1611-1728$	M_6
7	C_7	$344-360$	M_5	$1729-1792$	M_7
8	K_2	$361-384$	K_2	$1793-1800$	M_5
9	K_3	$385-392$	M_7	$1801-1920$	M_6
10	K_2	$393-400$	M_5	$1921-1960$	M_7
11	C_{11}	$401-420$	M_7	$1961-2048$	K_2
$12-14$	K_2	$421-432$	K_2	$2049-2058$	M_7
15	K_3	$433-448$	M_7	$2059-2100$	M_5
$16-20$	K_2	$449-450$	M_5	$2101-2240$	M_7
21	M_7	$451-454$	K_2	$2241-2304$	M_6
$22-24$	K_2	455	M_5	$2305-2352$	M_7
25	M_5	456	K_2	$2353-2400$	M_6
$26-27$	K_3	$457-460$	M_5	$2401-2560$	K_2
$28-40$	K_2	$461-512$	K_2	$2561-2744$	M_7
$41-42$	M_7	$513-525$	M_5	$2745-2800$	M_5
$43-45$	K_3	$526-560$	M_7	$2801-2880$	M_6
$46-48$	K_2	$561-576$	K_2	$2881-3072$	K_2
49	M_7	$577-588$	M_7	$3073-3136$	M_7
50	M_5	$589-600$	M_5	$3137-3200$	M_5
$51-64$	K_2	$601-640$	K_2	$3201-3456$	M_6
$65-70$	M_7	$641-686$	M_7	$3457-3584$	M_7
$71-80$	K_2	$687-720$	M_5	$3585-3840$	M_6
$81-84$	M_7	$721-768$	K_2	$3841-3920$	M_7
$85-96$	K_2	$769-784$	M_7	$3921-4096$	K_2
$97-98$	M_7	$785-800$	M_5	$4097-4116$	M_7
$99-100$	M_5	$801-840$	M_7	$4117-4200$	M_5
$101-105$	M_7	$841-864$	M_6	$4201-4320$	M_6
$106-108$	K_2	$865-896$	M_7	$4321-4480$	M_7
$109-112$	M_7	$897-900$	M_5	$4481-4608$	M_6
$113-128$	K_2	$901-912$	M_6	$4609-4704$	M_7
$129-140$	M_7	$913-920$	M_5	$4705-4800$	M_6
$141-144$	K_2	$921-936$	M_6	$4801-5120$	K_2
$145-147$	M_7	$937-940$	M_5	$5121-5184$	M_6
$148-150$	M_5	$941-960$	M_6	$5185-5488$	M_7
$151-160$	K_2	$961-980$	M_7	$5489-5600$	M_5
$161-168$	M_7	$981-1024$	K_2	$5601-5880$	M_6
$169-175$	M_5	$1025-1029$	M_7	$5881-5888$	K_2
$176-192$	K_2	$1030-1050$	M_5	$5889-5952$	M_6
$193-196$	M_7	$1051-1120$	M_7	$5953-6016$	K_2
$197-200$	M_5	$1121-1152$	M_6	$6017-6144$	M_6
$201-210$	M_7	$1153-1176$	M_7	$6145-6272$	M_7
$211-216$	K_2	$1177-1200$	M_5	$6273-6400$	M_5
$217-224$	M_7	$1201-1280$	K_2	$6401-6912$	M_6
225	M_5	$1281-1372$	M_7	$6913-7168$	M_7
$226-256$	K_2	$1373-1440$	M_5	$7169-7680$	M_6
$257-280$	M_7	$1441-1536$	K_2	$7681-7840$	M_7
$281-288$	K_2	$1537-1568$	M_7	$7841-8064$	M_6
$289-294$	M_7	$1569-1600$	M_5	$8065-8192$	K_2
$295-300$	M_5	$1601-1602$	M_6		

optimal algorithm is $A = M_7 K_2^2 C_7$, of total cost $\mathsf{M}_A(194) = 22 \cdot \mathsf{M}_{K_2^2 C_7}(28) + 3937 = 26575$ bit operations.

Figure 4 shows the recursive cost of the Karatsuba method, as used in Weimerskirch & Paar (2003), of our hybrid method, and the classical method.

Lemma 1 implies that the classical methods need only be considered for $n \leq 12$. We can prune $\mathcal{T}$ further and now illustrate this for K_3. One first checks that $M_{AK_3B}(3\mathsf{b}_A\mathsf{b}_B) < M_{K_3AB}(3\mathsf{b}_A\mathsf{b}_B)$ for $A \in \{K_2, M_5, M_6, M_7\}$, $B \in \mathcal{T}^*$, and $\mathsf{b}_B \geq 2$. Therefore for K_3 to be the top-level tool in an optimal algorithm over $\mathcal{T}$ the next algorithm to it must be either K_3 or C_b for some b. Since the classical method is not optimal for $n \geq 13$ and Table II does not list K_3 in the interval 46 to $3 \cdot 45 = 135$, K_3 is not the top-level tool for $n \geq 135$.

Table III gives the asymptotic behavior of the costs of the algorithms in the toolbox $\mathcal{T}$ when used recursively. It is expected that for very large polynomials only the asymptotically fastest method, namely M_6, should be used. But due to the tiny differences in the cost exponents this seems to happen only for very large polynomial lengths, beyond the sizes which are shown in Table II.

TABLE III

ASYMPTOTICAL COST $O(n^k)$ OF ALGORITHMS IN THE TOOLBOX $\mathcal{T}$

algorithm	k
C_b, $b \geq 2$	$\log_b b^2 = 2$
K_2	$\log_2 3 \approx 1.5850$
K_3	$\log_3 6 \approx 1.6309$
M_5	$\log_5 13 \approx 1.5937$
M_6	$\log_6 17 \approx 1.5812$
M_7	$\log_7 22 \approx 1.5885$

IV. HARDWARE STRUCTURE

The delay of a fully parallel combinational Karatsuba multiplier is $4\lceil \log_2 n \rceil$, which is almost 4 times that of a classical multiplier, namely $\lceil \log_2 n \rceil + 1$. It is the main disadvantage of the Karatsuba method for hardware implementations. Grabbe et al. (2003) suggested as solution a pipelined Karatsuba multiplier for 240-bit polynomials, shown in Figure 5.

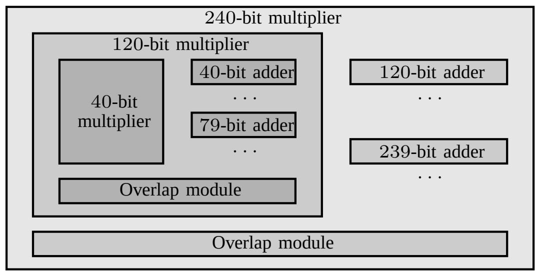


Fig. 5. The 240-bit multiplier by Grabbe et al. (2003)

The innermost part of the design is a combinational pipelined 40-bit classical multiplier equipped with 40-bit and 79-bit adders. The multiplier, these adders, and the overlap module, together with a control circuit, constitute a 120-bit multiplier. The algorithm is based on a modification of a Karatsuba formula for 3-segment polynomials. Another suitable control circuit performs the 2-segment Karatsuba method for 240 bits by means of a 120-bit recursion, 239-bit adders, and an overlap circuit.

We improve this multiplier with respect to both area and time. The multiplier of Grabbe et al. (2003) can be seen as implementing the factorization $240 = 2 \cdot 3 \cdot 40$. Table III implies that it is usually best to apply the 3-segment Karatsuba for small inputs. Translating this into hardware reality, the new design is based on the factorization $240 = 2 \cdot 2 \cdot 2 \cdot 30$.

The new 30-bit multiplier follows the recipe of Table II. It is a combinational circuit without feedback and the design goal is to minimize its area. In general, k pipeline stages can

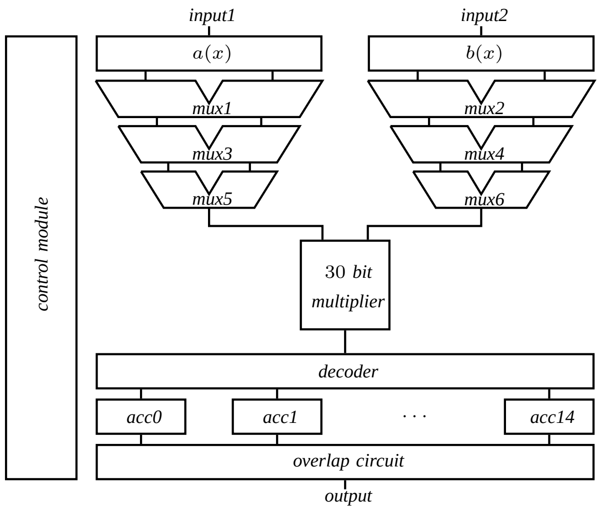


Fig. 6. The structure of the Karatsuba multiplier with few recursions

perform n parallel multiplications in $n + k - 1$ instead of nk clock cycles without pipelining.

In the recursive Karatsuba multiplier of Grabbe et al. (2003), the core of the system, namely the combinational multiplier, is idle for about half of the time. To improve the resource usage, we reduce the communication overhead by decreasing the levels of recursion. In this new 240-bit multiplier, an 8-segment Karatsuba is applied at once to 30-bit polynomials. We computed symbolically the formulas describing three recursive levels of Karatsuba, and implemented these formulas directly.

The new circuit is shown in Figure 6. The multiplexers *mux1* to *mux6* are adders at the same time. Their inputs are 30-bit sections of the two original 240-bit polynomials which are added according to the Karatsuba rules. Now their 27 output pairs are pipelined as inputs into the 30-bit multiplier. The 27 corresponding 59-bit polynomials are subsequently combined according to the overlap rules to yield the final result. The synchronization is set so that the 30-bit multipliers require 1 and 4 clock cycles for the classical and hybrid Karatsuba implementations, respectively.

The time and space consumptions after place and route are shown in Table IV and compared with the results of Grabbe et al. (2003) and the classical method. The second column shows the number of clock cycles for a multiplication. The third column represents the area in terms of number of slices. This measure contains both logic elements, or LUTs, and flip-flops used for pipelining. The fourth column is the multiplication time as returned by the hardware synthesis tool. Finally the last column shows the product of area and time in order to compare the AT measures of our designs.

TABLE IV
TIME AND AREA OF DIFFERENT 240-BIT MULTIPLIERS

multiplier type	**clock cycles**	**slices**	**time**	**AT** Slices $\times$ μs
classical	56	1582	$0.523\mu s$	827
the circuit of Fig. 5	54	1660	$0.655\mu s$	1087
hybrid (Fig. 6)	30	1480	$0.378\mu s$	559

V. CONCLUSION

In this paper we have shown how combining algorithmic techniques with platform dependent strategies can be used to develop designs which are highly optimized for FPGAs. These modules have been considered as appropriate implementation targets for cryptographic purposes both as prototyping platforms and as system on chips.

The benefits of hybrid implementations are well known for software implementations, where the crossover points between subquadratic and classical methods depend on the available memory and processor word size. There seems to be no previous systematic investigation on how to apply these methods efficiently for hardware implementations. We have shown that a hybrid implementation mixing classical and asymptotically fast recursive methods can result in significant area savings.

REFERENCES

[1] D. V. Bailey and C. Paar, "Optimal extension fields for fast arithmetic in public-key algorithms," in *Advances in Cryptology: Proceedings of CRYPTO '98*, Santa Barbara CA, ser. Lecture Notes in Computer Science, H. Krawczyk, Ed., no. 1462. Springer-Verlag, 1998, pp. 472–485.

[2] R. E. Blahut, *Fast Algorithms for Digital Signal Processing*. Reading MA: Addison-Wesley, 1985.

[3] D. G. Cantor, "On arithmetical algorithms over finite fields," *Journal of Combinatorial Theory, Series A*, vol. 50, pp. 285–300, 1989.

[4] *Digital Signature Standard (DSS)*, U.S. Department of Commerce / National Institute of Standards and Technology, January 2000, federal Information Processings Standards Publication 186-2.

[5] J. von zur Gathen and J. Gerhard, "Arithmetic and factorization of polynomials over $\mathbb{F}_2$," in *Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation ISSAC '96*, Zürich, Switzerland, Y. N. Lakshman, Ed. ACM Press, 1996, pp. 1–9.

[6] ——, *Modern Computer Algebra*, 2nd ed. Cambridge, UK: Cambridge University Press, 2003, first edition 1999.

[7] J. von zur Gathen and M. Nöcker, "Polynomial and normal bases for finite fields," *Journal of Cryptology*, vol. 18, no. 4, pp. 337–355, September 2005.

[8] J. von zur Gathen and J. Shokrollahi, "Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$," in *Selected Areas in Cryptography (SAC 2005)*. Springer-Verlag, 2005, to appear.

[9] C. Grabbe, M. Bednara, J. Shokrollahi, J. Teich, and J. von zur Gathen, "FPGA designs of parallel high performance $GF(2^{233})$ multipliers," in *Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS-03)*, vol. II, Bangkok, Thailand, May 2003, pp. 268–271.

[10] D. Hankerson, A. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*. Springer-Verlag, 2003.

[11] M. Jung, F. Madlener, M. Ernst, and S. Huss, "A Reconfigurable Coprocessor for Finite Field Multiplication in $GF(2^n)$," in *Workshop on Cryptographic Hardware and Embedded Systems*. Hamburg: IEEE, April 2002.

[12] A. Karatsuba and Y. Ofman, "Multiplication of multidigit numbers on automata," *Soviet Physics–Doklady*, vol. 7, no. 7, pp. 595–596, January 1963, translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July, 1962.

[13] Ç. K. Koç and S. S. Erdem, "Improved Karatsuba-Ofman Multiplication in $GF(2^m)$," US Patent Application, January 2002.

[14] P. L. Montgomery, "Five, Six, and seven-Term Karatsuba-Like Formulae," *IEEE Transactions on Computers*, vol. 54, no. 3, pp. 362–369, March 2005.

[15] C. Paar, "Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields," Ph.D. dissertation, Institute for Experimental Mathematics, University of Essen, Essen, Germany, June 1994.

[16] A. Weimerskirch and C. Paar, "Generalizations of the Karatsuba Algorithm for Efficient Implementations," Ruhr-Universität-Bochum, Germany, Tech. Rep., 2003.