Joachim von zur Gathen & Jamshid Shokrollahi (2005). Efficient FPGA-based Karatsuba multipliers for polynomials over F_2. In *Selected Areas in Cryptography (SAC 2005)*, Bart Preneel & Stafford Tavares, editors, number 3897 in Lecture Notes in Computer Science, 359–369. Springer-Verlag, Kingston, ON, Canada. ISBN 3-540-33108-5. URL https://dx.doi.org/10.1007/11693383.

Efficient FPGA-based Karatsuba multipliers for polynomials over $\mathbb{F}_2$

Joachim von zur Gathen and Jamshid Shokrollahi

B-IT, Görresstr. 13, Universität Bonn, 53113 Bonn, Germany
gathen@bit.uni-bonn.de
jamshid@bit.uni-bonn.de

Abstract. We study different possibilities of implementing the Karatsuba multiplier for polynomials over $\mathbb{F}_2$ on 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 the classical and the Karatsuba methods decreases the number of bit operations. On the other hand, a mixture of sequential and combinational circuit design techniques includes pipelining and 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.

Keywords: Finite field arithmetic, fast multiplication, asymptotically fast algorithms, Karatsuba method, hardware, FPGA.

1 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 [6]). 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 [3] and [4]). Traditional lore held that asymptotically fast methods are not suitable for hardware. We disprove this view in the present paper, continuing our work in [7].

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 [5], chapter 8, and Shoup's NTL software). The Karatsuba method has the lowest crossover point with the classical algorithm.

In hardware, the methods used are either platform independent or platform dependent. The first group consists of algorithmic optimizations which reduce the total number of operations, whereas the second approach uses specific properties of implementation environments to achieve higher performance.

The Karatsuba algorithm, for multiplication of large integers, was introduced in [10]. This algorithm is based on a formula for multiplying two linear polynomials which uses only 3 multiplications and 4 additions, as compared to 4 multiplications and 1 addition in the classical formula. The extra number of additions disappears asymptotically. This method can be applied recursively to 2^m-bit polynomials, where m is an integer. Here we optimize and adapt the Karatsuba algorithm for hardware realization of cryptographic algorithms.

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 [12], [1], [11], [8], chapter 2, and [5], chapter 8), but hardware implementations have attracted less attention. The only works known to us are [9], [14], and our previous paper [7]. [9] and [14] 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 those for the Karatsuba method. [7] contains a hybrid implementation of the Karatsuba method which reduces the latency by pipelining and by mixing sequential and combinational circuits.

The present work is to our knowledge the first one which tries to decrease the resource usage of polynomial multipliers using both known algorithmic and platform dependent methods. We present the best choice of hybrid multiplication algorithms for polynomials with at most 128 bits, as long as the choice is restricted to three (recursive) methods, namely classical, Karatsuba, and a variant of Karatsuba for quadratic polynomials. The "best" refers to minimizing the area measure. This is an algorithmic and machine independent optimization. In an earlier implementation ([7]) we had designed a 240-bit multiplier on a XC2V6000-4FF1517-4 FPGA. We re-use this structure to illustrate a second

type of optimization, which is machine-dependent. Our goal is a 240-bit multiplier with small area-time cost. This measure may be thought as the time on a single-bit processor. We now put a single 30-bit multiplier on our FPGA and use three Karatsuba steps to get from $240 = 2^3 \cdot 30$ to 30 bits. This requires judicious application of multiplexer and adder circuitry, but the major computational cost still resides in the multiplier. $27 = 3^3$ small multiplications are required for one 240-bit product, and these inputs are fed into the single small multiplier in a pipelined fashion. This has the pleasant effect of keeping the total delay small and the area reduced, with correspondingly small propagation delays. Using this 240-bit multiplier we cover in particular the 233-bit polynomials proposed by NIST for elliptic curve cryptography in [13].

One reviewer wrote: *The idea of using such a generalization of Karatsuba's method is not new, but it is usually dismissed for operands of relatively small sizes because of lower performance in software implementations. The fact that some area on an FPGA is saved is an interesting and new remark: the kind of remark usually "obvious"* **after** *one has seen it, but that only few seem able to see in the first place.*

The structure of this paper is as follows. First the Karatsuba method and its cost are studied in Section 2. Section 3 is devoted to optimized hybrid Karatsuba implementations. Section 4 shows how a hybrid structure and pipelining improves resource usage in our circuit from [7]. Section 5 analyzes the effect of the number of recursion levels on the performance, and Section 6 concludes the paper.

2 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:

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

We call this the 2-*segment Karatsuba method or* $\mathcal{K}_2$. 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.

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,

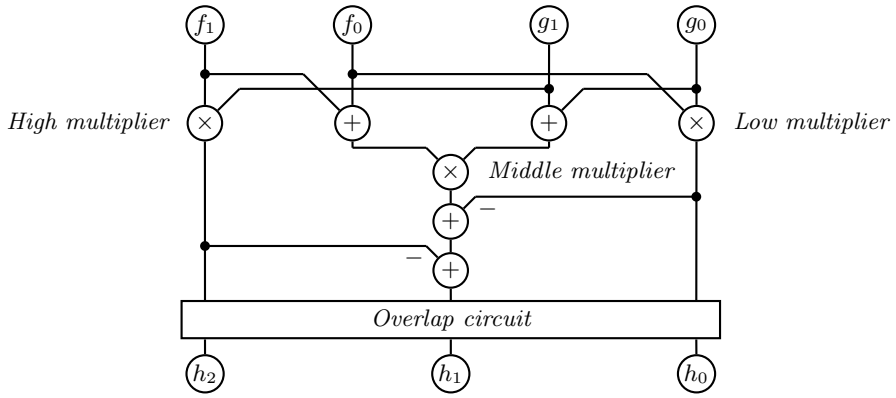

Fig. 1. The circuit to perform one level of the Karatsuba multiplication

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.

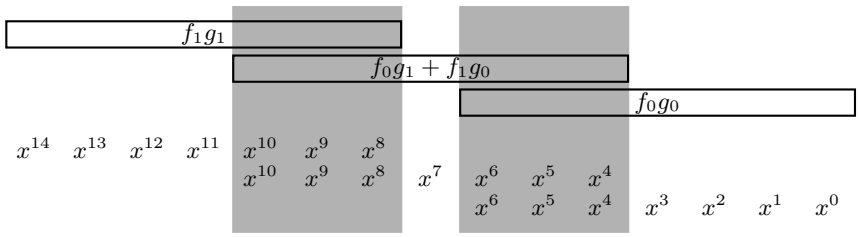

Fig. 2. The overlap circuit for the 8-bit Karatsuba multiplier

Figures 1 and 2 show that we need three recursive multiplication calls and some additions: $2m$ for input adders, $2(2m-1)$ for output adders, and $2(m-1)$ for the overlap module; where $m = \lceil n/2 \rceil$. If $\mathsf{M}_n^{(2)}$ is the total number of bit operations to multiply two n-bit polynomials, then

$$\mathsf{M}_n^{(2)} \leq 3\ \mathsf{M}_m^{(2)} + 8m - 4. \tag{2}$$

When n is a power of 2, with the initial values of $\mathsf{M}_1^{(2)} = 1$ we get:

$$\mathsf{M}_n^{(2)} \leq 7\ \lceil n^{\log_2 3} \rceil - 8n + 2. \tag{3}$$

The gain in Karatsuba's method is visually illustrated in Figure 8.2 of [5]. The delay of the circuit for $n \geq 2$ is at most

$$4\lceil \log_2 n \rceil \tag{4}$$

times the delay of a single gate. On the other hand, a classical multiplier for n-bit polynomials requires

$$2n^2 - 2n + 1 \tag{5}$$

gates and has a propagation delay of

$$1 + \lceil \log_2 n \rceil. \tag{6}$$

To multiply two quadratic polynomials, we use the following formula from [2] which we call 3-*segment Karatsuba or* $\mathcal{K}_3$. It uses 6 multiplications and 12 additions when used for fields of characteristic 2, compared to 9 multiplications and 4 additions in the classical method:

$$\begin{aligned}
&(a_2x^2 + a_1x + a_0)(b_2x^2 + b_1x + b_0) = \\
&a_2b_2x^4 + ((a_1 + a_2)(b_1 + b_2) - a_1b_1 - a_2b_2)x^3 \\
&+ ((a_2 + a_0)(b_2 + b_0) - a_0b_0 + a_1b_1 - a_2b_2)x^2 \\
&+ ((a_1 + a_0)(b_1 + b_0) - a_0b_0 - a_1b_1)x + a_0b_0.
\end{aligned} \tag{7}$$

Similar to (2) we can write the recursive costs of $\mathcal{K}_3$ as:

$$\mathsf{M}_n^{(3)} \leq 6\ \mathsf{M}_m^{(3)} + 22m - 10, \tag{8}$$

where $m = \lceil n/3 \rceil$.

Since $\log_2 3 \approx 1.5850 < 1.6309 \approx \log_3 6$, this approach is asymptotically inferior to the original Karatsuba method. One result of this paper is to determine the range of usefulness for this method (namely some $n \leq 81$) on our type of hardware.

3 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 [5], chapter 8, and [8], chapter 2). The corresponding issues for hardware implementations have not been discussed in the literature, except that our previous paper [7] uses classical multipliers for polynomials with up to 40 bits.

We present a general methodology and execute it in the special case of a toolbox with these algorithms: classical, $\mathcal{K}_2$, and $\mathcal{K}_3$. The general idea is that we have a toolbox $\mathcal{A}$ of recursive multiplication algorithms. Each algorithm $A \in \mathcal{A}$ computes the product of two polynomials of degree less than n, for any n. The cost of A consists in some arithmetic operations plus recursive multiplications. For simplicity, we assume that the *optimal hybrid multiplication routine using* A is built from the bottom up. For each $n \geq 1$, we determine the best method for n-bit polynomials, starting with a single arithmetic operation (namely, a multiplication) for constant polynomials ($n = 1$). For $n \geq 2$, we compute the cost of applying each $A \in \mathcal{A}$ to n-bit polynomials, using the already computed

optimal values for the recursive calls. We then enter into our table one of the algorithms with minimal cost.

We now execute this general approach on our toolbox $\mathcal{A} = \{\text{classical}, \mathcal{K}_2, \mathcal{K}_3\}$. The costs are given in (2) and (8). Whenever necessary, polynomials are padded with leading zeros. The results are shown in Table 1.

The first column gives the number n of bits, so that we deal with polynomials of degree up to $n-1$. The second column "rec" specifies the first recursive level, that is the algorithm from $\mathcal{A} = \{\text{classical}, \mathcal{K}_2, \mathcal{K}_3\}$ to be used, abbreviated as $\{C, 2, 3\}$. The column "cost" gives the total number of arithmetic operations. The next column states the "ratio" of practice to theory, namely $c \cdot \text{cost}/n^{\log_2 3}$, where the constant c is chosen so that the last entry is 1. The asymptotic regime visibly takes over already at the fairly small values that we consider. The final column gives the cost of algorithm from [14], which is Karatsuba-based. We know of no other implementation that can be easily compared with ours.

For example, the entry $n = 41$ refers to polynomials of degree up to 40. The entry 2 in column "$\mathcal{A}$" says that $\mathcal{K}_2$ is to be employed at the top of the recursion. Since $m = \lceil 41/2 \rceil = 21$, (2) says that three pairs of 21-bit polynomials need to be multiplied, plus $8 \cdot 21 - 4 = 164$ operations. One has to look up the algorithm for 21 bits in the table. Continuing in this way, the prescription for 41 bits is:

n	41	21	7
algorithm	$\mathcal{K}_2$	$\mathcal{K}_3$	C
add	164	144	85

$$\text{total} = 164 + 3 \cdot (144 + 6 \cdot 85) = 2126.$$

In the recursive call of $\mathcal{K}_2$ at $n = 41$, the inputs are split into two pieces of 20 and 21 bits. It is tempting to single out one of the three recursive multiplications as a 20-bit operation, and indeed this view is taken in [14]. They pad input polynomials with enough zero coefficients and apply the Karatsuba method in a recursive manner. Operations involving a coefficient known to be zero are neglected. In our designs, we use three 21-bit multiplications, for a small loss in the operations count but a huge gain in modularity: we only implement a single 21-bit multiplier, thus simplifying the design and enabling pipelining. Section 4 exemplifies this (with 30 rather than 21 bits).

We note that designers of fast arithmetic software have used the general methodology sketched above, in particular formulating it as breakpoint between different algorithms. The classical algorithm can also be viewed recursively, which is used for some results in Table 2 below.

The goal of our hybrid design is to minimize the total arithmetic cost. The same methodology can, of course, also be applied to multi-objective applications, say minimizing A and AT. A concern with them would be to limit the number of table entries that are kept.

4 Hardware structure

According to (4) and (6), the delay of a fully parallel combinational Karatsuba multiplier is almost 4 times that of a classical multiplier. It is the main disad-

Table 1. The number of operations for the hybrid method for polynomial degrees below 128, and Karatsuba's algorithm according to [14]

length	hybrid			Karatsuba	length	hybrid			Karatsuba
	rec	cost	ratio			rec	cost	ratio	
3	C	13	0.404	19	66	2	4886	1.131	5402
4	C	25	0.492	33	67	2	4894	1.106	5675
5	C	41	0.567	61	68	2	4894	1.081	5812
6	2	59	0.611	77	69	2	4926	1.063	6091
7	C	85	0.689	110	70	2	4926	1.039	6231
8	2	103	0.676	127	71	2	4934	1.017	6374
9	3	134	0.730	175	72	2	4934	0.995	6041
10	2	159	0.733	219	73	2	5713	1.127	6737
11	C	221	0.875	257	74	2	5713	1.103	6883
12	2	221	0.763	275	75	2	5721	1.081	7032
13	2	307	0.933	346	76	2	5721	1.059	7107
14	2	307	0.830	382	77	2	5753	1.043	7262
15	3	346	0.838	421	78	2	5753	1.022	7340
16	2	369	0.807	441	79	2	5761	1.003	7421
17	2	470	0.934	572	80	2	5761	0.983	7381
18	2	470	0.853	593	81	3	6536	1.094	7777
19	2	553	0.921	707	82	2	6702	1.100	7935
20	2	553	0.849	733	83	2	6710	1.080	8096
21	3	654	0.930	817	84	2	6710	1.060	8177
22	2	747	0.986	855	85	2	7528	1.167	8344
23	2	755	0.929	896	86	2	7528	1.146	8428
24	2	755	0.869	917	87	2	7536	1.126	8515
25	3	992	1.070	1064	88	2	7536	1.106	8559
26	3	992	1.005	1138	89	2	7544	1.087	8738
27	3	992	0.947	1215	90	2	7544	1.068	8828
28	2	1029	0.927	1254	91	2	7699	1.071	8921
29	2	1154	0.984	1337	92	2	7699	1.053	8968
30	2	1154	0.932	1379	93	2	7731	1.039	9067
31	2	1231	0.944	1424	94	2	7731	1.022	9117
32	2	1231	0.898	1447	95	2	7739	1.006	9170
33	2	1542	1.071	1714	96	2	7739	0.989	9197
34	2	1542	1.021	1848	97	2	9904	1.245	9800
35	2	1550	0.981	1985	98	2	9904	1.225	10102
36	2	1550	0.938	2054	99	2	9912	1.207	10407
37	2	1807	1.047	2197	100	2	9912	1.188	10560
38	2	1807	1.003	2269	101	2	9944	1.173	10871
39	2	1815	0.967	2344	102	2	9944	1.155	11027
40	2	1815	0.929	2355	103	2	9952	1.138	11186
41	2	2126	1.047	2537	104	2	9952	1.121	11266
42	2	2126	1.007	2615	105	2	9984	1.107	11589
43	3	2396	1.094	2696	106	2	9984	1.091	11751
44	3	2396	1.055	2737	107	2	9992	1.075	11916
45	3	2396	1.018	2824	108	2	9992	1.060	11999
46	2	2445	1.003	2868	109	2	10357	1.082	12170
47	2	2453	0.973	2915	110	2	10357	1.067	12256
48	2	2453	0.941	2939	111	2	10365	1.053	12345
49	2	3172	1.177	3238	112	2	10365	1.038	12390
50	2	3172	1.140	3388	113	2	11522	1.137	12737
51	2	3180	1.108	3541	114	2	11522	1.122	12911
52	2	3180	1.074	3618	115	2	11530	1.107	13088
53	2	3188	1.045	3777	116	2	11530	1.092	13177
54	2	3188	1.014	3857	117	2	11562	1.080	13360
55	2	3307	1.022	3940	118	2	11562	1.066	13452
56	2	3307	0.993	3982	119	2	11570	1.052	13547
57	2	3690	1.078	4153	120	2	11570	1.038	13595
58	2	3690	1.048	4239	121	2	12295	1.089	13790
59	2	3698	1.022	4328	122	2	12295	1.075	13888
60	2	3698	0.996	4373	123	2	12303	1.062	13989
61	2	3937	1.033	4468	124	2	12303	1.048	14040
62	2	3937	1.006	4516	125	2	12335	1.038	14147
63	2	3945	0.983	4567	126	2	12335	1.025	14201
64	2	3945	0.959	4593	127	2	12343	1.013	14258
65	2	4886	1.159	5132	128	2	12343	1.000	14288

vantage of the Karatsuba method for hardware implementations. In [7], we have suggested as solution a pipelined Karatsuba multiplier for 240-bit polynomials, shown in Figure 3.

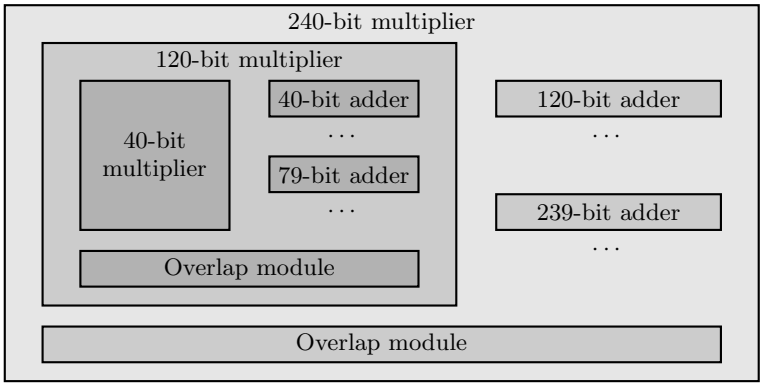

Fig. 3. The 240-bit multiplier in [7]

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 which is similar to but slightly different from (7). (We were not aware of this better formula at that time.)

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.

This multiplier can be seen as implementing the factorization $240 = 2 \cdot 3 \cdot 40$. Table 1 implies that it is usually best to apply the 2-segment Karatsuba, except for small inputs. Translating this into hardware reality, we now present a better design based on the factorization $240 = 2 \cdot 2 \cdot 2 \cdot 30$. The resulting structure is shown in Figure 4.

The 30-bit multiplier follows the recipe of Table 1. It is a combinational circuit without feedback and the design goal was to minimize its area. In general, k pipeline stages can perform n parallel multiplications in $n + k - 1$ instead of nk clock cycles without pipelining.

We have implemented our design, the structure of [7], and a purely classical implementation, on an XC2V6000-4FF1517-4 FPGA. The classical design has a classical 30-bit multiplier and applies the three classical recursion steps. The results after place and route are shown in Table 2. 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

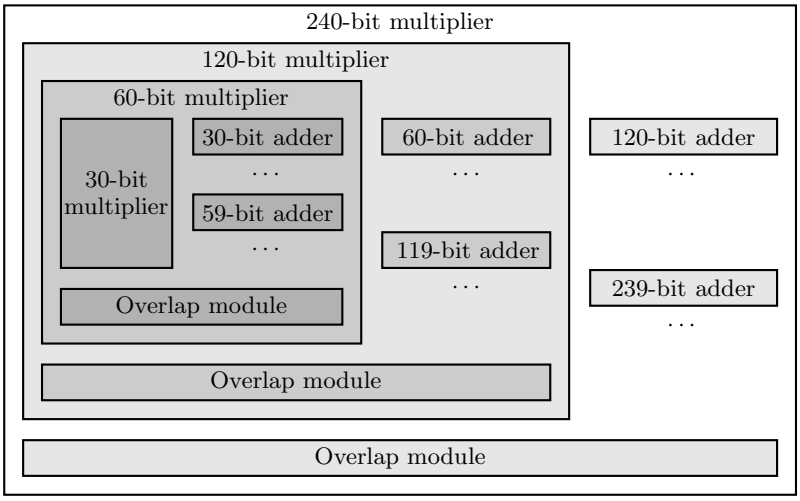

Fig. 4. The new 240-bit multiplier

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.

The synchronization is set so that the 30-bit multipliers require 1 and 4 clock cycles for classical and hybrid Karatsuba implementations, respectively. The new structure is smaller than the implementation in [7] but requires more area than the classical one. This drawback is due to the complicated structure of the Karatsuba method but is compensated by speed as seen in the time and AT measures. In the next section we further improve our structure by decreasing the number of recursions.

Table 2. Time and area of different multipliers for 240-bit polynomials

Multiplier type	**Number of clock cycles**	**Number of slices**	**Multiplication time**	**AT** Slices $\times$ μs
classical	106	1328	$1.029\mu s$	1367
The circuit of [7] (Fig. 3)	54	1660	$0.655\mu s$	1087
Hybrid Karatsuba (Fig. 4)	55	1513	$0.670\mu s$	1014

5 Hybrid polynomial multiplier with few recursions

In the recursive Karatsuba multiplier of [7], the core of the system, namely the combinational multipliers, is idle for about half of the time. To improve 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 5. 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. Time and space consumptions are shown in Table 3 and compared with the results of [7]. The columns are as in Table 2. We see that this design improves on the previous ones in all respects.

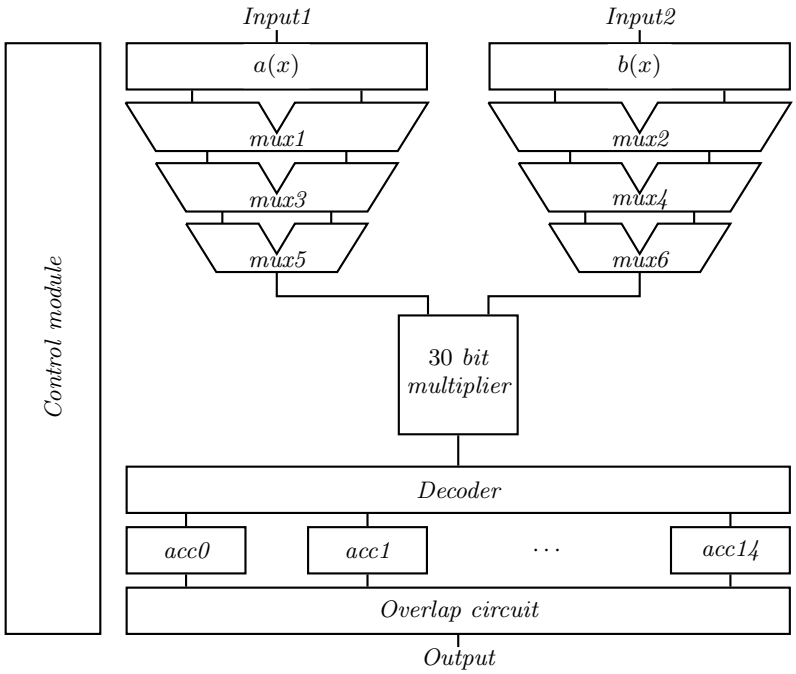

Fig. 5. The structure of the Karatsuba multiplier with fewer number of recursions

Table 3. Time and area of different 240-bit multipliers with reduced number of recursion levels

Multiplier type	**Number of clock cycles**	**Number of slices**	**Multiplication time**	**AT** Slices $\times$ μs
classical	56	1582	$0.523\mu s$	827
The circuit of [7](Fig. 3)	54	1660	$0.655\mu s$	1087
Hybrid Karatsuba (Fig. 5)	30	1480	$0.378\mu s$	559

6 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.

We improved the structure proposed in [7] in both time and area aspects. The time has been improved by decreasing the number of recursion stages. To minimize the area we have further improved the results of [14], as witnessed in Table 1, by applying the Karatsuba method in a hybrid manner. 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. In this paper we have shown that a hybrid implementation mixing classical and two Karatsuba methods can result in significant area savings.

Comparisons with the work of [7] are shown in Table 3. The asymptotic methods are better than classical multipliers both with respect to time and area measures. An obvious open question is to optimize a larger class of recursive algorithms than our $\mathcal{K}_2$ and $\mathcal{K}_3$.

7 Acknowledgements

We thank Roberto Avanzi for various pointers to the literature and for pointing out to us the formula from [2].

References

1. Bailey, D.V., Paar, C.: Optimal extension fields for fast arithmetic in public-key algorithms. In Krawczyk, H., ed.: Advances in Cryptology: Proceedings of CRYPTO '98, Santa Barbara CA. Number 1462 in Lecture Notes in Computer Science, Springer-Verlag (1998) 472–485
2. Blahut, R.E.: Fast Algorithms for Digital Signal Processing. Addison-Wesley, Reading MA (1985)
3. Cantor, D.G.: On arithmetical algorithms over finite fields. Journal of Combinatorial Theory, Series A **50** (1989) 285–300
4. von zur Gathen, J., Gerhard, J.: Arithmetic and factorization of polynomials over $\mathbb{F}_2$. In Lakshman, Y.N., ed.: Proceedings of the 1996 International Symposium on Symbolic and Algebraic Computation ISSAC '96, Zürich, Switzerland, ACM Press (1996) 1–9 Technical report tr-rsfb-96-018, University of Paderborn, Germany, 1996, 43 pages. Final version in Mathematics of Computation.
5. von zur Gathen, J., Gerhard, J.: Modern Computer Algebra. Second edn. Cambridge University Press, Cambridge, UK (2003) First edition 1999.

6. von zur Gathen, J., Nöcker, M.: Polynomial and normal bases for finite fields. Journal of Cryptology (2005) to appear.
7. Grabbe, C., Bednara, M., Shokrollahi, J., Teich, J., von zur Gathen, J.: FPGA designs of parallel high performance $GF(2^{233})$ multipliers. In: Proc. of the IEEE International Symposium on Circuits and Systems (ISCAS-03). Volume II., Bangkok, Thailand (2003) 268–271
8. Hankerson, D., Menezes, A., Vanstone, S.: Guide to Elliptic Curve Cryptography. Springer-Verlag (2003)
9. Jung, M., Madlener, F., Ernst, M., Huss, S.: A Reconfigurable Coprocessor for Finite Field Multiplication in $GF(2^n)$. In: Workshop on Cryptographic Hardware and Embedded Systems, Hamburg, IEEE (2002)
10. Karatsuba, A., Ofman, Y.: Multiplication of multidigit numbers on automata. Soviet Physics–Doklady **7** (1963) 595–596 translated from Doklady Akademii Nauk SSSR, Vol. 145, No. 2, pp. 293–294, July, 1962.
11. Koç, Ç.K., Erdem, S.S.: Improved Karatsuba-Ofman Multiplication in $GF(2^m)$. US Patent Application (2002)
12. Paar, C.: Efficient VLSI Architectures for Bit-Parallel Computation in Galois Fields. PhD thesis, Institute for Experimental Mathematics, University of Essen, Essen, Germany (1994)
13. U.S. Department of Commerce / National Institute of Standards and Technology: Digital Signature Standard (DSS). (2000) Federal Information Processings Standards Publication 186-2.
14. Weimerskirch, A., Paar, C.: Generalizations of the karatsuba algorithm for efficient implementations. Technical report, Ruhr-Universität-Bochum, Germany (2003)