General Digit-Serial Normal Basis Multiplier with Distributed Overlap

Martin Novotný, Jan Schmidt
Department of Computer Science and Engineering, CTU FEE in Prague
Karlovo nám. 13, 121 35 Praha 2, Czech Republic
{novotnym, schmidt}@fel.cvut.cz

Abstract

We present the architecture of digit-serial normal basis multiplier over GF(2^m). The multiplier was derived from the multiplier of Agnew et al. Proposed multiplier is scalable by the digit width of general value in difference of the multiplier of Agnew et al. that may be scaled only by digit width that divides the degree m. This helps designers to trade area for speed e.g. in public-key cryptographic systems based on elliptic-curves, where m should be a prime number. Functionality of multiplier has been tested by simulation and implemented in Xilinx Virtex 4 FPGA.

1. Introduction

Arithmetic operations over finite fields find many application areas including cryptography and error-correcting codes. The finite fields are of the characteristic p (p is prime) and degree m, marked $GF(p^m)$. We focus on the subset $GF(2^m)$, where all field elements are expressed as m-bit vectors. While addition of two elements in $GF(2^m)$ is performed as a bit-wise XOR operation, multiplication of two elements is more complicated. An algorithm for multiplication depends on representation of field elements, which is given by chosen basis. The most common kinds are polynomial basis and normal basis [3].

Normal basis [7] offers some advantages and has some disadvantages in comparison with the polynomial basis. On the minus side, field degrees m usable for cryptography are rare as they must be prime [8] and should have a so called optimal normal basis. On the plus side, multiplication and squaring are simpler even for general field degree. Thereby arithmetic units implementing operations over finite field in normal basis are smaller and faster, that leads to better performance/area ratio in comparison with systems that operate on polynomial basis [9]. Therefore, they attract continuous attention [4].

One of indicators of quality of the system is the ability to trade the area for speed, that we call *scalability* in this paper. This ability is important e.g. in cryptographic systems, where the multiplier is one of units repeatedly used during execution of cryptographic algorithm. As the multiplication is time-consuming operation, it may represent a bottle-neck of the algorithm. The scalability of multiplier allows the designer to improve the performance of cryptographic system by acceleration of multiplication. This acceleration is realized at the cost of larger area of multiplier, but, as the multiplier is a part of larger system, even the quality factor (performance/area ratio) of the whole system may increase.

Finite field multipliers may be scaled by digit width. While the polynomial basis multiplier can be scaled by any digit width D, the normal basis multiplier developed by Agnew et al. [1] can be scaled well only by digit width D that divides the degree m. [8]. As m must be prime for cryptographic purposes, the standard normal basis multiplier cannot be scaled at all. This disqualifies normal basis multiplier in comparison with polynomial basis multiplier.

Our aim was to adapt the architecture of a normal basis multiplier to be scalable by any digit width D independently of degree m. In this paper we introduce and compare two different architectures of general digit-serial normal basis multipliers that we developed.

The paper is structured as follows: In the second chapter, we bring information about normal basis multiplication and describe the standard digit-serial normal basis multiplier. In the third chapter, we introduce new architecture of general digit-serial multiplier, where the digit width does not divide the degree m. In the fourth chapter, we discuss the area and performance of multiplier in terms of gate count and critical path length. Finally, in the fifth chapter, we present results of experiments comparing the new multiplier with the standard solution.

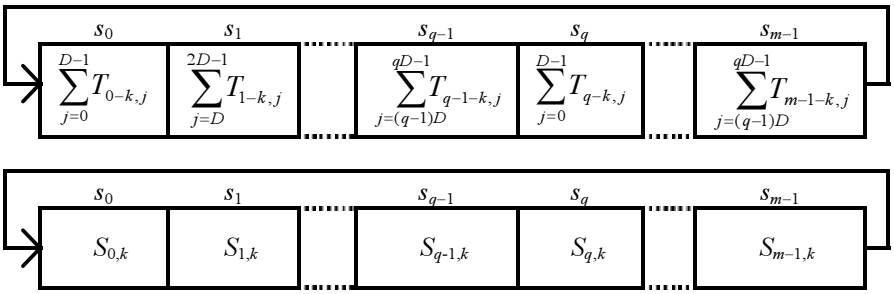

Figure 1. Evaluation of terms in register *C* in standard digit-serial multiplier, a) full notation, b) abbreviated notation

2. Preliminaries

2.1. Mathematical background

Let a and b be elements of $GF(2^m)$ with normal basis. Multiplication of two elements in $GF(2^m)$ can be defined by multiplication matrix λ with entries $\lambda_{ij} \in GF(2)$ [3]. The coefficients of the product $c = a \times b$ are

$$c_i = \sum_{j=0}^{m-1} a_{j+i} \sum_{l=0}^{m-1} b_{l+i} \lambda_{jl}\ ,\ i = 0 \ldots m-1, \qquad (1)$$

where additions and multiplications are performed in $GF(2)$. The indices of a and b are added modulo m.

The volume of hardware in the multiplier is given by the number C_N of non-zero entries in λ. A normal basis multiplier can be scaled by digit-serialization, that is, D bits (called a *digit*) are evaluated in one clock cycle. Pipelined multipliers evaluate D terms in every stage of the multiplier in one clock cycle. An overview of normal basis multipliers can be found e. g. in [2]. Here we deal with the pipelined Massey-Omura multiplier on which the general digit-width multipliers are based.

2.2. Pipelined digit-serial Massey-Omura multiplier

Agnew, et al. [1] modified the Massey-Omura multiplier [6] by pipelining and parallelization. The digit-serial version of this multiplier for digit width D that divides the degree m can be simply derived. It contains 3 registers. Registers A and B hold arguments a and b. The product $c = a \times b$ is successively evaluated in register C. All registers are rotated one bit right between cycles.

From (1) it follows that the equation for each bit of result can be divided into m terms $T_{i,j}$:

$$c_i = T_{i,0} + T_{i,1} + \ldots + T_{i,m-2} + T_{i,m-1}, \qquad (2)$$

where

$$T_{i,j} = a_{j+i} \sum_{l=0}^{m-1} b_{l+i} \lambda_{jl}$$

The logic in front of each C register bit implements D of the terms $T_{i,j}$. We shall call this logic together with the register bit a *stage*.

Rule 1 (digit-serial multiplier): Let $q = m/D$ be the number of clock cycles of one multiplication. Then in the k-th clock cycle ($\forall k \in \langle 0, q-1 \rangle$), the stage s_r ($\forall r \in \langle 0, m-1 \rangle$) evaluates the set of terms

$$S_{r,k} = \sum_{j=u}^{v} T_{r-k,j}\ , \qquad (3)$$

where $u = rD \bmod m$, $v = u + D - 1$

(The values of subscript indices are reduced modulo m.)

The set of terms $S_{r,k}$ is added to the partial result of the bit c_{r-k}, which is, due to the rotation of register C, present in the stage s_r during the k-th clock cycle. The result $c_0 c_1 c_2 \ldots c_{m-1}$ is available in stages $s_{q-1} s_q s_{q+1} \ldots s_{m-1} s_0 \ldots s_{q-2}$ after q clock cycles.❐

Let the block of q consecutive stages be denoted as a *pipeline block*. From (3) it follows that any pipeline block implements exactly m terms $T_{i,j}$. The whole multiplier implements exactly $D \cdot m$ terms $T_{i,j}$, hence, we may split it into D pipeline blocks. From (3) also follows that terms $T_{i,j}$ evaluated in stages s_i and s_{i+q} have the same second indices j.

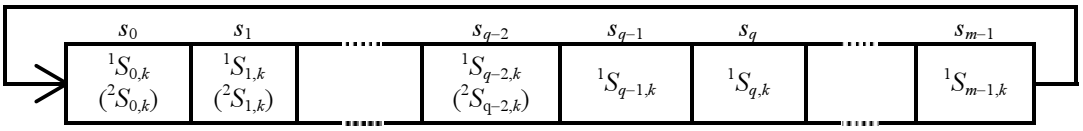

Figure 2a. Evaluation of terms in stages of register C in circular digit-serial multiplier GCCONC

s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
0,1 (10)	2,3 (0,1)	4,5 (2,3)	6,7 (4,5)	8,9 (6,7)	10	0,1	2,3	4,5	6,7	8,9

Figure 2b. Evaluation of terms in stages of circular digit-serial multiplier GCCONC for $m = 11$ and $D = 2$. Values of second indices j of terms $T_{i,j}$ are introduced. Multiplication takes $q = 6$ clock cycles.

s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}
0,1,2 (9,10)	3,4,5 (0,1,2)	6,7,8 (3,4,5)	9,10	0,1,2	3,4,5	6,7,8	9,10	0,1,2	3,4,5	6,7,8

Figure 2c. Evaluation of terms in stages of circular digit-serial multiplier GCCONC for $m = 11$ and $D = 3$. Values of second indices j of terms $T_{i,j}$ are introduced. Multiplication takes $q = 4$ clock cycles.

3. General digit-serial multiplier

Construction of the standard digit-serial multiplier is possible only for digit widths D that divide degree m. On the other hand, as stated before, m should be prime number for cryptographic purposes. Therefore, the standard digit-serial multipliers cannot be used in cryptographic systems. Our aim was to adapt the digit-serial multiplier for such cases. We developed two architectures for digit-serial multiplier that can be scaled by *any* digit width. These two architectures we call *circular multiplier with concentrated overlap (GCCONC)* and *(optimized) circular multiplier with distributed overlap (GCDIST and GCDO)*. The letter *G* stands for *general*, as the multipliers can be constructed for general value of digit width, in contrast to standard multiplier that can be constructed only if the digit width D divides the total number of bits m.

3.1. Circular multiplier with concentrated overlap (GCCONC)

In Figure 2 we describe a structure of a circular multiplier, concretely how the terms are evaluated in stages of register C. To get the correct result, stages $s_0 \dots s_{q-2}$ must be able to evaluate two different groups of terms. When holding partial results of any of bits $c_0 \dots c_{q-2}$, they must evaluate the first group (symbolized as ${}^1S$), but when holding partial results of any of bits $c_{m-q+1} \dots c_{m-1}$, they must evaluate the second group (${}^2S$). The stages are step-by-step switched from the group ${}^1S$ to the group ^{2}S during the computation as partial results of bits $c_{m-q+1} \dots c_{m-1}$ successively move to stages $s_0 \dots s_{q-2}$. In the k-th clock cycle, stages $s_k \dots s_{q-2}$ evaluate groups ${}^1S$, while stages $s_0 \dots s_{k-1}$ are switched to evaluate groups ${}^2S$. Stages $s_{q-1} \dots s_{m-1}$ do not switch and evaluate groups ${}^1S$ during the whole computation.

A shift register can be used to control successive switching of groups. The shift register is initially cleared; during computation series of '1's is step-by-step shifted in it. Evaluation of the result takes $q = \lceil m/D \rceil$ clock cycles.

Rule 2 (circular digit-serial multiplier with concentrated overlap GCCONC):
Let $q = \lceil m/D \rceil$ be the number of clock cycles of one multiplication. Then in the k-th clock cycle ($\forall k \in \langle 0, q-1 \rangle$) the stage s_r ($\forall r \in \langle 0, m-1 \rangle$) evaluates the set of terms $S_{r,k}$:

a) if $r \geq k$:

$$S_{r,k} = {}^1S_{r,k} = \sum_{j=u}^{v} T_{r-k,j}, \tag{4}$$

where

$$u = rD \bmod qD,$$
$$v = \min\{u + D - 1;\ m-1\}$$

b) if $r < k$:

$$S_{r,k} = {}^2S_{r,k} = \sum_{j=u}^{v} T_{r-k,j},$$

where

$$u = (r+m)D \bmod qD,$$
$$v = \min\{u + D - 1;\ m-1\}$$

(The values of subscript indices are reduced mod m)

The set of terms $S_{r,k}$ is added to the partial result of the bit c_{r-k}, which is, due to the rotation of register C, present in the stage s_r during the k-th clock cycle. The result $c_0c_1c_2\ldots c_{m-1}$ is available in stages $s_{q-1}s_qs_{q+1}\ldots s_{m-1}s_0\ldots s_{q-2}$ after q clock cycles. ❐

Example 1:
Let $m = 11$. It is shown in Figures 2b and 2c how the terms are evaluated in stages of register C in circular digit-serial multiplier. Indices in parentheses belong to sets of terms ${}^2S$.

The circular multiplier drawback lies in fact that $q-1$ pipeline stages must evaluate two different sets of logic. In other words, the circular multiplier contains almost $D+1$ pipeline blocks (instead of D pipeline blocks in case of standard multiplier) that overlap in $q-1$ stages (hence we call it the multiplier with *concentrated overlap*). The multiplier must implement almost $(D+1)\cdot m$ terms $T_{i,j}$. Moreover, we need $q-1$ multiplexers that also consume relatively large amount of area and may lie on critical path. Also, the necessity of successive switching of groups may slightly complicate the control.

3.2. Circular multiplier with distributed overlap (GCDIST and GCDO)

The main idea of this multiplier is to distribute the overlap, make it smaller and simplify the control. The number of terms evaluated in this multiplier is the same like in the standard one, i.e. $D\cdot m$ terms $T_{i,j}$. The area overhead insists in less than $m/2$ AND gates in comparison with the standard multiplier.

As stated before, in case of standard multiplier we may split the multiplier into D pipeline blocks, each containing q consecutive stages. All D pipeline blocks together create the whole multiplier, as $D \times q = m$. This rule can be satisfied only if D divides m. For such D that do not divide m the value $q = \lceil m/D \rceil$ and for these cases $D \times q > m$.

To divide the multiplier into D pipeline blocks as equally as possible, some of pipeline blocks must contain q pipeline stages, while some other must contain $q-1$ stages only. Exactly, (m mod D) pipeline blocks will contain q stages (mentioned later as *long pipeline blocks*), while the remaining (D – (m mod D)) pipeline blocks will contain $q-1$ stages (mentioned later as *short pipeline blocks*).

The idea of this multiplier is to compute all m terms $T_{i,j}$ not in q stages, but only in $q-1$ stages of each pipeline block. If the pipeline block is q stages long, then one of the stages will be empty (nothing will be computed in this stage). If the block is only $q-1$ stages long (short pipeline block), then first $q-1$ stages of the following block will be switched-off in the last clock cycle (or, equivalently, the last $q-1$ stages of previous pipeline block will be switched-off in the first clock cycle) to compute the empty stage. Switching-off some stages is less hardware complex than multiplexing between two different sets of logic. While the multiplexer is relatively complex, the AND gate used for switching-off is relatively simple. Evaluation of the result takes again $q = \lceil m/D \rceil$ clock cycles and cannot be shortened – the empty stage appears at different clock cycles for different partial results c_i.

There are plenty of possible definitions of this multiplier. One of them may assume that the first (m mod D) pipeline blocks are q stages long (long blocks), succeeded by (D – (m mod D)) pipeline blocks being $q-1$ stages long (short blocks). For such assumption we bring the following description.

Rule 3 (circular digit-serial multiplier with distributed overlap (GCDIST)):
Let $q = \lceil m/D \rceil$ be the number of clock cycles of one multiplication. Let $D_x = \lceil m/(q-1) \rceil$ be the maximum number of terms being evaluated in one stage. Let $F = ((m \bmod D) \times q)$ be the number of stages of all long pipeline blocks. Then in the k-th clock cycle ($\forall k \in \langle 0, q-1 \rangle$) the stage s_r ($\forall r \in \langle 0, m-1 \rangle$) evaluates the set of terms $S_{r,k}$:

(*long pipeline blocks*):
a) if $\big((r < F) \wedge (r \bmod q = 0)\big)$:

$$S_{r,k} = \varnothing$$

b) if $\big((r < F) \wedge (r \bmod q \neq 0)\big)$:

$$S_{r,k} = \sum_{j=u}^{v} T_{r-k,j}, \qquad (5)$$

where

$$u = (r-1 \bmod q) \times D_x,$$
$$v = \min\{u + D_x - 1;\ m-1\}$$

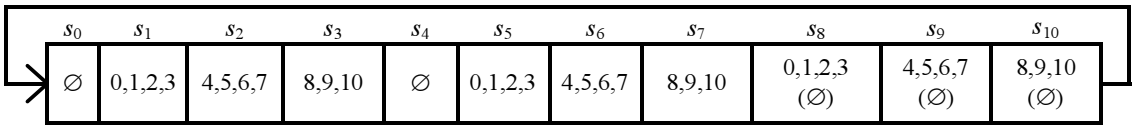

Figure 3a. Evaluation of terms in stages of circular digit-serial multiplier with distributed overlap (GCDIST as well as GCDO) for $m = 11$ and $D = 3$. Values of second indices j of terms $T_{i,j}$ are introduced. Multiplication logic is switched-off in stages s_8 through s_{10} in the last clock cycle (denoted by ∅). Multiplication takes $q = 4$ clock cycles.

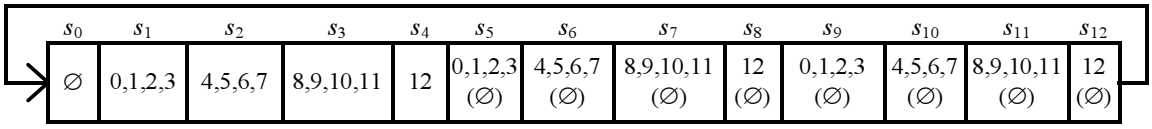

Figure 3b. Evaluation of terms in stages of circular digit-serial multiplier with distributed overlap (GCDIST) for $m = 13$ and $D = 3$. Values of second indices j of terms $T_{i,j}$ are introduced. Multiplication logic is switched-off in stages s_5 through s_{12} in the last clock cycle (denoted by ∅). Multiplication takes $q = 5$ clock cycles.

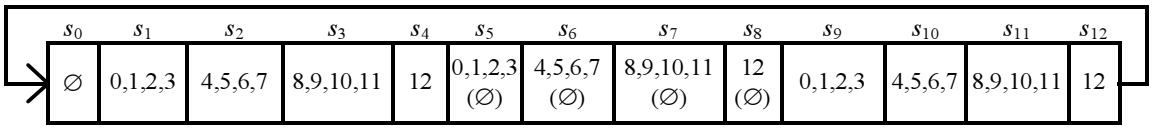

Figure 3c. Evaluation of terms in stages of optimized circular digit-serial multiplier with distributed overlap (GCDO) for $m = 13$ and $D = 3$. Values of second indices j of terms $T_{i,j}$ are introduced. Multiplication logic is switched-off in stages s_5 through s_8 in the first and last clock cycle (denoted by ∅). Multiplication takes $q = 5$ clock cycles.

(*short pipeline blocks*):

c) if $((r \geq F) \wedge (k = q-1))$:

$$S_{r,k} = \varnothing$$

d) if $((r \geq F) \wedge (k \neq q-1))$:

$$S_{r,k} = \sum_{j=u}^{v} T_{r-k,j},$$

where

$$u = ((r-F) \bmod (q-1)) \times D_x,$$

$$v = \min\{u + D_x - 1;\ m-1\}$$

❐

This multiplier evaluates the same amount of terms like the standard multiplier. The area overhead lies in the number of AND gates that are necessary to switch-off some stages in the last clock cycle. In the worst case, the number of the necessary AND gates could be almost m.

But, yet another area improvement is possible. As mentioned above, to implement the "empty stage" in short pipeline blocks, we can either switch-off the first $q-1$ stages of the following block in the last clock cycle or switch-off the last $q-1$ stages of the previous block in the first clock cycle. Fortunately, we can join these two approaches together. We do not need to switch-off all short blocks in the last (or first) clock cycle. Instead of that we can switch-off only all *odd* short blocks in the first *and* the last clock cycle, while *even* blocks will not be switched at all. This approach minimizes hardware resources even more. The number of additional AND gates is consequently less than $m/2$.

Rule 4 (optimized circular digit-serial multiplier with distributed overlap (GCDO)):

Let $q = \lceil m/D \rceil$ be the number of clock cycles of one multiplication. Let $D_x = \lceil m/(q-1) \rceil$ be the maximum number of terms being evaluated in one stage. Let $F = ((m \bmod D) \times q)$ be the number of stages of all

long pipeline blocks. Then in the k-th clock cycle ($\forall k \in \langle 0, q-1 \rangle$) the stage s_r ($\forall r \in \langle 0, m-1 \rangle$) evaluates the set of terms $S_{r,k}$:

(*long pipeline blocks*):

a) if $(r < F) \wedge (r \bmod q = 0)$:

$$S_{r,k} = \varnothing$$

b) if $(r < F) \wedge (r \bmod q \neq 0)$:

$$S_{r,k} = \sum_{j=u}^{v} T_{r-k,j},$$

where

$$u = (r-1 \bmod q) \times D_x,$$
$$v = \min\{u + D_x - 1;\ m-1\}$$

(*short pipeline blocks*):

c) if $(r \geq F) \wedge \left(\left(\left\lfloor \frac{r-F}{q-1} \right\rfloor \bmod 2 = 0 \right) \wedge ((k=0) \vee (k=q-1)) \right)$ (6)

$$S_{r,k} = \varnothing$$

d) if $(r \geq F) \wedge not \left(\left(\left\lfloor \frac{r-F}{q-1} \right\rfloor \bmod 2 = 0 \right) \wedge ((k=0) \vee (k=q-1)) \right)$

$$S_{r,k} = \sum_{j=u}^{v} T_{r-k,j},$$

where

$$u = (r-F \bmod (q-1)) \times D_x,$$
$$v = \min\{u + Dx - 1;\ m-1\}$$

❐

Example 2:
Let $m = 11$ and $D = 3$. It is shown in Figure 3a how the terms are evaluated in stages of register C in GCDIST and GCDO multipliers. Coincidently, in this case, the multipliers contain only one short pipeline block and hence the structure of both multipliers is the same.

Table 1. Hardware resources

	AND gates	XOR gates	MUX
standard	mD	$\leq C_N D$	0
GCCONC	$< mD + m$	$< C_N D + C_N$	$q-2$
GCDO	$< mD + m/2$	$\leq C_N D$	0

Table 2. Critical path length

	AND	XOR	MUX
standard	1	$\leq \lceil \log_2 \lceil C_N/m \rceil \rceil + \lceil \log_2 D \rceil$	0
GCCONC	1	$\leq \lceil \log_2 \lceil C_N/m \rceil \rceil + \lceil \log_2 D \rceil$	≤ 1
GCDO	1-2	$\leq \lceil \log_2 \lceil C_N/m \rceil \rceil + \lceil \log_2 D \rceil + 1$	0

The difference between GCDIST and GCDO multipliers is illustrated in Figures 3b and 3c. The multipliers are constructed for $m = 13$ and $D = 3$. The multipliers contain two short pipeline blocks. While both short pipeline blocks are equipped with switching-off AND gates in the GCDIST multiplier, only one short pipeline block is equipped with AND gates in GCDO multiplier.

4. Area and critical path length

In the standard digit-serial multiplier (D divides m), every stage evaluates exactly D terms; consequently, the entire logic of the multiplier evaluates mD terms. Every term is implemented by one AND gate and at most $\lceil C_N/m \rceil$ XOR gates, which form a tree with depth $\leq \lceil \log_2 \lceil C_N/m \rceil \rceil$. Terms evaluated in one stage are summed up by another tree of XOR gates with depth $\lceil \log_2 D \rceil$.

The circular multiplier with concentrated overlap (GCCONC) contains additional logic that in the first $q-1$ stages evaluates alternative set of terms. Each set contains at most D terms. First $q-1$ stages contain multiplexers that may lie on critical path. Moreover, we need also additional control logic for successive switching the first $q-1$ stages. This control logic may be implemented either by shift register or by the counter with decoder.

In circular multiplier with distributed overlap (GCDIST and GCDO) each stage may evaluate up to $2D$ terms. But, we must note that this is a marginal case. Some stages contain one AND gate to switch-off all logic in a stage. This AND gate may lie on the critical path. The total number of additional AND gates is less than m in case of GCDIST or less than $m/2$ in case of GCDO.

In Table 1 we compare hardware resources necessary for the standard digit-serial multiplier, the circular multiplier with concentrated overlap GCCONC and optimized circular multiplier with distributed overlap GCDO. The comparison of critical path lengths is in Table 2.

5. Implementation results

We implemented the above multipliers in Xilinx Virtex 4 VLX25 FPGA with Leonardo Spectrum 2005 and ISE 7.1i. We measured the area of the design as the number of slices used and observed the minimum clock period. The time of multiplication is a product of the minimum clock period and the number of clock cycles. The quality factor is a reciprocal of the time-area product.

To retrieve the knowledge of the overhead of the GCCONC and GCDO multipliers against the standard multiplier we performed one set of experiments with composite degree m. For these experiments we have chosen $m = 180$, that has rich set of divisors. Hence we can construct the standard multiplier for relatively large amount of digit widths D. For such Ds that do not divide m the theoretical (but unreachable) results of the standard multiplier can be interpolated. Since the structure of the GCCONC multiplier is the same as the structure of the standard multiplier when the digit widths D divides the degree m (no switching between two sets of logic is necessary in this case), we executed another set of experiments for a prime number $m = 173$. In this set, we compared the architectures of GCCONC and GCDO multipliers only.

Implementation results for the first twenty values of D are summarized in Tables 3 and 4. The dependencies of multiplication time on digit width D for composite degree $m = 180$ and for prime degree $m = 173$ are available in Figure 4. Recall that results for the standard multiplier and Ds not dividing the degree m are not available since standard multiplier cannot be constructed for such cases.

The dependencies of quality factor on digit width D are available in Figure 5. The quality factor here is evaluated as performance/area ratio of multiplier alone. It decreases with growing D as the achievable clock frequency decreases for larger Ds. But, when using multiplier e.g. in cryptographic system, the critical path may lay in other units outside the multiplier. For such cases, the quality factor of the whole system may increase with growing D unless the critical path transfers to the multiplier.

The implementation results correspond to theoretical values introduced in Tables 1 and 2. They confirm that the general digit-serial multipliers are of the same quality as the standard one. In other words, we do not pay much for the added flexibility.

The standard multiplier still remains the best solution whenever D divides m. If the digit width D does not divide degree m, e.g. when m is prime number, the GCCONC or GCDO multiplier may be chosen. As we expected, GCDO provides better results in both multiplication time and area, hence leading to better quality factor.

6. Conclusions

We have presented two architectures of general digit-serial normal basis multipliers. The multipliers are useful in applications where the acceleration of normal basis multiplication is required and the digit-width D generally does not divide the field degree m. Such applications primarily include cryptographic algorithms where m should be a prime number.

Our implementation shows that, in practical circumstances, the overhead of the proposed multipliers over standard multiplier of Agnew et. al. is small and the units are suitable for general use. The strategy presented here is applicable also to other types of pipelined normal basis multipliers, e.g. [4] and [5].

Table 3. Implementation results for m = 180

	Mult. Time [µs]			Area [slices]			Quality factor		
D	std	CONC	GCDO	std	CONC	GCDO	std	CONC	GCDO
1	0.80	0.82	0.90	365	365	455	3.44	3.35	2.43
2	0.49	0.49	0.57	455	455	508	4.45	4.48	3.46
3	0.45	0.44	0.44	640	643	643	3.44	3.53	3.51
4	0.34	0.38	0.40	736	735	757	3.95	3.57	3.29
5	0.34	0.32	0.36	912	910	912	3.19	3.44	3.05
6	0.31	0.29	0.33	997	1013	1045	3.24	3.37	2.86
7		0.30	0.30		1223	1113		2.77	3.03
8		0.27	0.27		1370	1272		2.71	2.89
9	0.25	0.23	0.25	1361	1368	1411	2.98	3.17	2.82
10	0.22	0.21	0.23	1451	1458	1494	3.10	3.32	2.89
11		0.22	0.21		1699	1634		2.62	2.91
12	0.18	0.19	0.21	1725	1731	1768	3.14	3.10	2.68
13		0.18	0.18		1921	1863		2.84	3.02
14		0.17	0.17		2088	1967		2.75	2.94
15	0.16	0.16	0.16	2079	2081	2128	3.02	3.02	2.97
16		0.16	0.16		2054	2216		3.07	2.80
17		0.16	0.16		2379	2343		2.69	2.71
18	0.13	0.14	0.15	2442	2442	2485	3.06	3.01	2.70
19		0.15	0.17		2419	2589		2.75	2.33
20	0.13	0.15	0.13	2713	2712	2712	2.85	2.54	2.74

Table 4. Implementation results for m = 173

	Period [ns]		Mul. time [µs]		Area [slices]		Quality factor	
D	CONC	GCDO	CONC	GCDO	CONC	GCDO	CONC	GCDO
1	4.86	4.93	0.84	0.85	352	438	3.38	2.67
2	7.41	6.13	0.64	0.53	619	487	2.51	3.85
3	10.86	8.23	0.63	0.48	725	615	2.19	3.41
4	13.15	8.23	0.58	0.36	815	724	2.12	3.81
5	9.83	8.98	0.34	0.31	989	872	2.94	3.65
6	12.19	11.07	0.35	0.32	1100	969	2.57	3.21
7	12.05	11.03	0.30	0.28	1173	1068	2.83	3.40
8	12.61	11.61	0.28	0.26	1319	1215	2.73	3.22
9	14.02	12.23	0.28	0.24	1396	1340	2.56	3.05
10	14.15	12.53	0.25	0.23	1475	1432	2.66	3.10
11	14.24	12.73	0.23	0.20	1661	1557	2.64	3.15
12	14.41	13.02	0.22	0.20	1729	1669	2.68	3.07
13	14.02	13.56	0.20	0.19	1781	1778	2.86	2.96
14	13.16	14.39	0.17	0.19	1946	1914	3.00	2.79
15	13.36	15.09	0.16	0.18	2058	2003	3.03	2.76
16	14.86	13.99	0.16	0.15	2176	2135	2.81	3.04
17	16.71	15.19	0.18	0.17	2219	2260	2.45	2.65
18	14.42	14.61	0.14	0.15	2382	2342	2.91	2.92
19	14.99	14.05	0.15	0.14	2345	2474	2.84	2.88
20	17.51	16.15	0.16	0.15	2614	2576	2.43	2.67

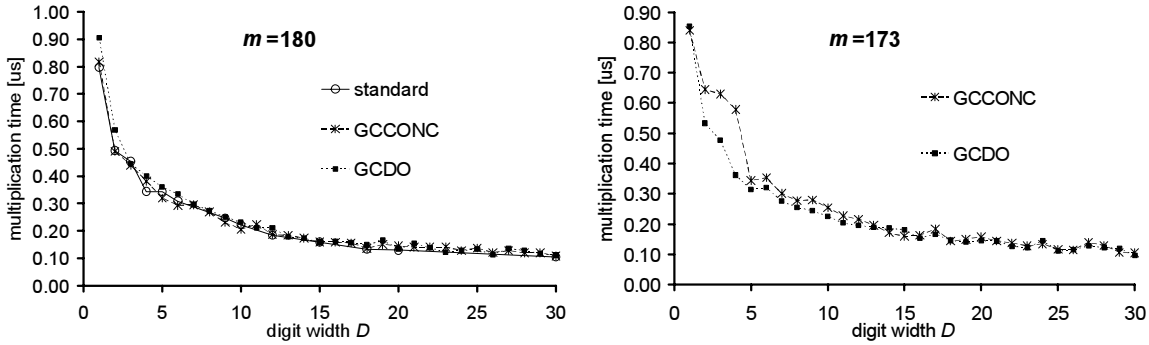

Figure 4. Time spent for calculation of one product for variable digit widths

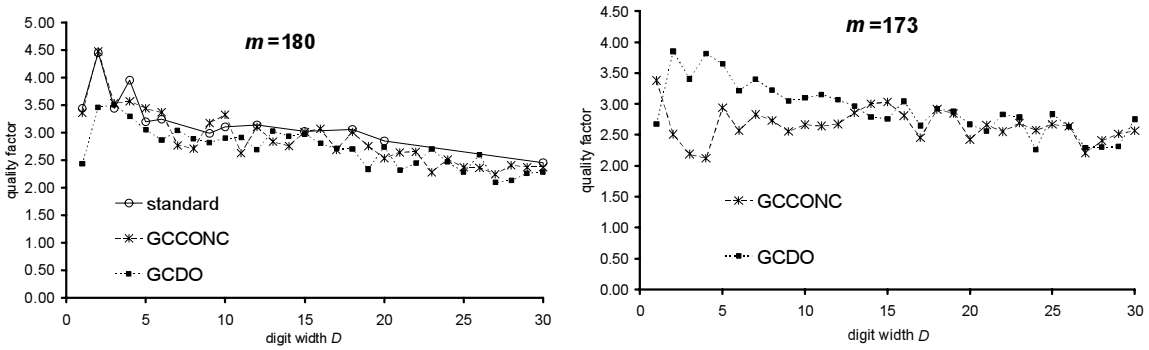

Figure 5. Quality factor as a function of digit width

7. References

[1] Agnew, G.B, Mullin. R.C., Onyszchuk, I.M. and Vanstone, S.A.: An Implementation for a Fast Public-Key Cryptosystem. Journal of Cryptology vol. 3, pp.63-79, 1991

[2] Ahlquist, G. C., Nelson, B., Rice, M. : Optimal Finite Field Multipliers for FPGAs. In: Proceedings of the 9th International Workshop on Field Programmable Logic and Applications (FPL'1999), Springer-Verlag, LNCS 1673, pp 51-60, 1999.

[3] IEEE 1363 Standard for Public-key Cryptography, 2000

[4] Kwon, S., Gaj, K., Kim, C. H. and Hong, P.C.: Efficient Linear Array for Multiplication in GF(2m) Using a Normal Basis for Elliptic Curve Cryptography, In: CHES 2004, Springer-Verlag, LNCS 3156, pp. 76-91, 2004

[5] Reyhani-Masoleh, A., Hasan, M.A.: Low Complexity Sequential Normal Basis Multipliers over GF(2m), In: 16th IEEE Symposium on Computer Arithmetic (ARITH-16 '03), pp. 188-195, IEEE 2003

[6] Massey, J. and Omura, J.: Computational Method and Apparatus for Finite Field Arithmetic. U.S. patent number 4,587,627, 1986

[7] Mullin, R., Onyszchuk, I., Vanstone, S., Wilson, R.: Optimal Normal Bases in GF(p^n). Discrete Applied Mathematics, vol. 22, pp. 149-161, 1989

[8] NIST, "Digital Signature Standard," *FIPS Publication*, 186-2, 2000

[9] Schmidt, J., Novotný, M., Jäger, M., Bečvář, M., Jáchim, M.: Exploration of Design Space in ECDSA, In: Proceedings of FPL 2002 (LNCS2438), pp. 1072-1075, Springer-Verlag, 2002

Acknowledgement

This work has been supported by the MSM6840770014 research program (Research in the Area of the Prospective Information and Navigation Technologies)