Multiplication over $\mathbb{F}_{p^m}$ on FPGA: A Survey*

Jean-Luc Beuchat, Takanori Miyoshi, Yoshihito Oyama, and Eiji Okamoto

Laboratory of Cryptography and Information Security
University of Tsukuba
1-1-1 Tennodai, Tsukuba
Ibaraki, 305-8573, Japan

Abstract. This paper aims at comparing multiplication algorithms over $\mathbb{F}_{p^m}$ on FPGA. Contrary to previous surveys providing the reader with an estimate of both area and delay in terms of XOR gates, we discuss place-and-route results which point out that the choice of an algorithm depends on the irreducible polynomial and on some architectural parameters. We designed a VHDL code generator to easily study a wide range of algorithms and parameters.

1 Introduction

Multiplication over $\mathbb{F}_{p^m}$ is a fundamental calculation in elliptic curve cryptography (ECC), pairing-based cryptography, and implementation of error-correcting codes. Field programmable gate arrays (FPGAs) have become more and more popular to implement hardware accelerators for such algorithms. This paper aims at comparing several multiplication algorithms in order to help engineers and researchers in selecting the most appropriate architecture for a given application on FPGAs.

Some papers survey multiplication over $\mathbb{F}_{p^m}$ from a theoretical point of view and provide the reader with an estimate of both area and delay of hardware operators in terms of XOR gates (see for instance [4, 6]). However, such results do not include routing delays and seem difficult to apply to FPGAs, whose architecture is usually based on look-up tables (LUTs). We propose here a comparison based on place-and-route results on Xilinx Spartan-3 FPGAs. In order to easily modify the irreducible polynomial and some architectural parameters (e.g. pipeline stages), we designed a tool which generates a structural VHDL description, as well as scripts to automatically place-and-route the operator and collect significant results.

There are several ways to encode elements of an extension field. In this paper, we will only consider the well-known polynomial representation. Let $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ be a degree-m monic irreducible polynomial over $\mathbb{F}_p$, where p is a prime. Then, $\mathbb{F}_{p^m} = \mathbb{F}_p[x]/f(x)$, and an element $a(x) \in \mathbb{F}_{p^m}$ is a degree-$(m-1)$ polynomial with coefficients in $\mathbb{F}_p$: $a(x) = a_{m-1}x^{m-1} + \ldots + a_1 x +$

* This work was supported by the New Energy and Industrial Technology Development Organization (NEDO), Japan.

P.C. Diniz et al. (Eds.): ARC 2007, LNCS 4419, pp. 214–225, 2007.

a_0. Three families of algorithms allow one to compute $a(x)b(x) \bmod f(x)$ (where $a(x)$ and $b(x)$ belong to $\mathbb{F}_{p^m}$). In parallel-serial schemes, a single coefficient of the multiplier $a(x)$ is processed at each step. This leads to small operands performing a multiplication in m steps. Parallel multipliers compute a degree-$(2m-2)$ polynomial and carry out a final modular reduction. They achieve a higher throughput at the price of a larger circuit area. Song and Parhi introduced array multipliers as a trade-off between computation time and circuit area [12]. The idea consists in processing D coefficients of the multiplier at each step. The parameter D is sometimes referred to as *digit size* and parallel-serial schemes can be considered as a special case with $D = 1$.

This paper focuses on array multipliers, which are often preferred to parallel architectures for hardware implementation of ECC or pairing-based cryptography [7, 5, 10, 11, 3]. Depending on the order in which coefficients of $a(x)$ are processed, multiplication modulo $f(x)$ can be performed according to two schemes: most-significant element (MSE) first and least-significant element (LSE) first. Sections 2 and 3 are respectively devoted to the study of MSE first and LSE first algorithms. We discuss experiment results in Section 4.

2 Most-Significant Element (MSE) First Algorithms

The celebrated Horner's rule allows the design of parallel-serial algorithms starting with the most-significant element (MSE). The simplest scheme requires m iterations to compute $p(x) = (a(x)b(x)) \bmod f(x)$. Recall that the equation

$$a(x)b(x) = ((\ldots(a_{m-1}b(x)x + a_{m-2}b(x))x + \ldots)x + a_1b(x))x + a_0b(x)$$

can easily be computed recursively. A register stores the intermediate result $p(x)$ which is initially set to zero. At step i, $m-1 \geq i \geq 0$, we compute $p(x) = xp(x)+a_ib(x)$. Note that $a_ib(x)$ is a degree-$(m-1)$ polynomial. Thus, it suffices to reduce $xp(x)$ at each step to perform a modulo $f(x)$ multiplication[1] (Algorithm 1 and Figure 1a).

Algorithm 1. MSE multiplication over $\mathbb{F}_{p^m}$.

Require: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$.
Ensure: $p(x) = a(x)b(x) \bmod f(x)$
1: $p(x) \leftarrow 0$;
2: **for** i from $m-1$ downto 0 **do**
3: $\quad p(x) \leftarrow a_ib(x) + (xp(x)) \bmod f(x)$;
4: **end for**

Several researchers proposed to process D coefficients of operand $a(x)$ at each clock cycle in order to reduce the computation time. Shu *et al.* introduced for

[1] Another implementation of this algorithm was proposed in [8]. Since it involves more hardware and a more complex control, we will not consider it in our experiments.

instance an algorithm which performs multiplication over $\mathbb{F}_{p^m}$ in $\lceil m/D \rceil$ clock cycles [11]. At step i, they compute a sum of D partial products reduced modulo $f(x)$:

$$t(x) = \sum_{j=0}^{D-1} a_{Di+j}(x^j b(x) \bmod f(x)),$$

where $t(x)$ is a degree-$(m-1)$ polynomial. A second degree-$(m-1)$ polynomial $p(x)$ accumulates these partial products. Algorithm 2 summarizes this multiplication scheme (see also Figure 1b).

Algorithm 2. MSE multiplication over $\mathbb{F}_{p^m}$ [11].

Require: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$. The algorithm requires a degree-$(m-1)$ polynomial $t(x)$ for intermediate computations.
Ensure: $p(x) = a(x)b(x) \bmod f(x)$
1: $p(x) \leftarrow 0$;
2: **for** i from $\lceil m/D \rceil - 1$ downto 0 **do**
3: $\quad t(x) \leftarrow \sum_{j=0}^{D-1} a_{Di+j}(x^j b(x) \bmod f(x))$;
4: $\quad p(x) \leftarrow t(x) + (x^D p(x) \bmod f(x))$;
5: **end for**

Song and Parhi suggested to compute at each step a degree-$(m + D - 2)$ polynomial $t(x)$ which is the sum of D partial products [12]:

$$t(x) = \sum_{j=0}^{D-1} a_{Di+j} x^j b(x). \tag{1}$$

A degree-$(m+D-1)$ polynomial $s(x)$, updated according to Horner's rule, allows to accumulate these partial products:

$$s(x) = t(x) + x^D(s(x) \bmod f(x)).$$

After $\lceil m/D \rceil$ iterations, $s(x)$ is a degree-$(m+D-1)$ polynomial congruent with $a(x)b(x)$ modulo $f(x)$. Song and Parhi included specific hardware to perform a modular correction [12]. A second approach, which does not require extra resources, was proposed in [3]. The idea consists in performing an additional iteration with $a_{-j} = 0$, $1 \leq j \leq D$. Since $t(x)$ is now equal to zero, we obtain $s(x) = x^D((a(x)b(x)) \bmod f(x))$ and

$$p(x) = s(x)/x^D = a(x)b(x) \bmod f(x).$$

Therefore, the m most-significant coefficients of $s(x)$ give the result (Algorithm 3 and Figure 1c). Note that a single modulo $f(x)$ reduction is needed, whereas Algorithm 2 requires D modular operations. Furthermore, from purely theoretical

point of view, the critical path is shorter than the one of an operator based on Algorithm 2. Assume for instance that $p = 3$ and that elements of $\mathbb{F}_3$ are encoded with two bits. Addition and multiplication of two elements can thus be performed by means of four-input tables. On FPGAs featuring four-input LUTs, generation and addition of D partial products require $\lceil \log_2 D \rceil + 1$ stages of LUTs. The algorithm by Shu *et al.* involves a number of additional stages which depends on D and $f(x)$. However, since partial products are not reduced in Algorithm 3, their addition requires more hardware resources (degree up to $m + D - 2$ instead of $m - 1$). Our experiment results suggest that the choice between Algorithms 2 and 3 depends on D and $f(x)$ (Section 4).

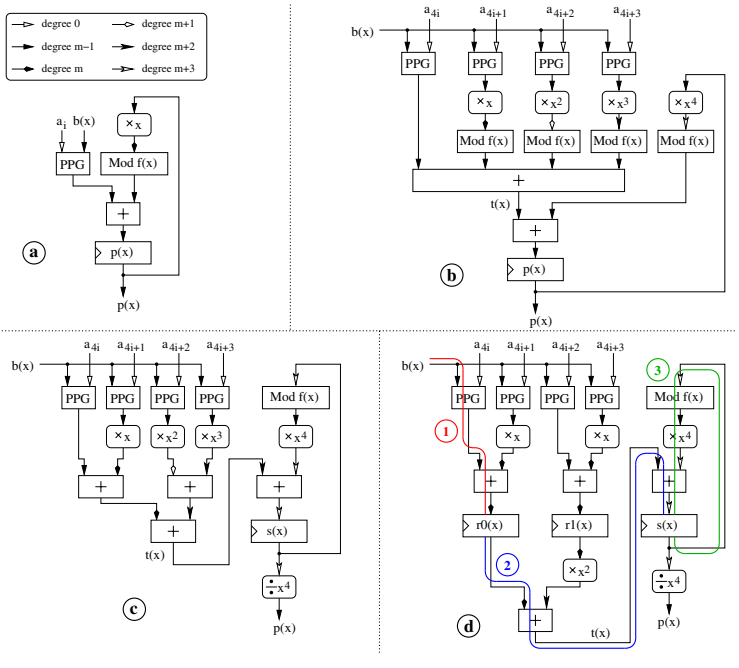

Fig. 1. MSE multiplication over $\mathbb{F}_{p^m}$. (a) Horner's rule. (b) Algorithm proposed by Shu *et al.* [11] ($D = 4$). (c) Algorithm proposed by Song and Parhi [12, 3] ($D = 4$). (d) Algorithm proposed by Song and Parhi with a pipeline stage ($D = 4$ and $R = 2$). Boxes with rounded corners involve only wiring.

Pipelining the computation of $t(x)$ (Equation (1)) sometimes allows one to shorten the critical path of a multiplier based on Algorithm 3 (Figure 1d). The depth of the pipeline stage is specified by a parameter R which indicates the number of registers inserted. Thus, $s(x)$ is the sum of $(R + 1)$ polynomials. Since the irreducible polynomial $f(x)$ is known at design time, all degree-$(m-1)$

Algorithm 3. MSE multiplication over $\mathbb{F}_{p^m}$ [3].

Require: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$. We assume that $a_{-j} = 0$, $1 \leq j \leq D$. The algorithm requires a degree-$(m+D-1)$ polynomial $s(x)$ as well as a degree-$(m+D-2)$ polynomial $t(x)$ for intermediate computations.

Ensure: $p(x) = a(x)b(x) \bmod f(x)$

1: $s(x) \leftarrow 0$;
2: **for** i from $\lceil m/D \rceil - 1$ downto -1 **do**
3: $\quad t(x) \leftarrow \sum_{j=0}^{D-1} a_{Di+j} x^j b(x)$;
4: $\quad s(x) \leftarrow t(x) + x^D \cdot (s(x) \bmod f(x))$;
5: **end for**
6: $p(x) \leftarrow s(x)/x^D$;

polynomials $x^i \bmod f(x)$, $m \leq i \leq m+D-1$, can be precomputed and the modulo $f(x)$ reduction of $s(x)$ is defined as follows:

$$s(x) \bmod f(x) = \sum_{i=0}^{m-1} s_i x^i + \sum_{i=m}^{m+D-1} s_i (x^i \bmod f(x)). \tag{2}$$

Both area and delay of a circuit implementing Equation (2) depend on D, $f(x)$, and FPGA family. Consider for example an FPGA embedding four-input LUTs, $\mathbb{F}_{3^{11}} = \mathbb{F}_3[x]/(x^{11} + 2x^8 + 1)$, and $D = 3$. Since $x^{11} \bmod f(x) = x^8 + 2$, $x^{12} \bmod f(x) = x^9 + 2x$, and $x^{13} \bmod f(x) = x^{10} + 2x^2$, we obtain

$$\sum_{i=11}^{13} s_i \cdot (x^i \bmod f(x)) = s_{13}x^{10} + s_{12}x^9 + s_{11}x^8 + 2s_{13}x^2 + 2s_{12}x + 2s_{11}.$$

Recall that, if each element $s_i \in \mathbb{F}_3$ is represented in radix-2 by two bits y_1 and y_0 (i.e. $s_i = 2y_1 + y_0$), multiplication by two is achieved by swapping y_0 and y_1. Therefore, the above equation involves only wiring and Equation (2) requires a single addition over $\mathbb{F}_{3^m}$. However, if we select $f(x) = x^{11} + x^{10} + x^8 + 2x^7 + x + 1$, we have:

$$\begin{aligned} x^{11} &= 2x^{10} + 2x^8 + x^7 + 2x + 2, \\ x^{12} &= x^{10} + 2x^9 + 2x^8 + 2x^7 + 2x^2 + 1, \\ x^{13} &= x^{10} + 2x^9 + x^8 + x^7 + 2x^3 + 2, \end{aligned}$$

and Equation (2) requires $D = 3$ adders over $\mathbb{F}_3$ to compute the degree-$(m-1)$ coefficient of the result. Thus, depending on $f(x)$, the number of operands ranges from 2 to $D+1$. Since four-input tables efficiently carry out addition over $\mathbb{F}_3$, the critical path of the circuit implementing Equation (2) contains up to $\lceil \log_2(D+1) \rceil$ LUTs. Consider now the operator described by Figure 1d. We need to study three paths to decide whether pipelining is efficient or not:

1. In characteristic three, computing a coefficient of a partial product requires a single LUT. Therefore, the longest path from input to pipeline stage includes $\lceil \log_2 \lceil D/R \rceil \rceil + 1$ LUTs.

2. Since computation of $s(x)$ involves $R+1$ operands, the path between the pipeline stage and the accumulator contains $\lceil \log_2(R+1) \rceil$ LUTs.
3. The accumulation loop includes a modulo $f(x)$ reduction and a single addition. The number of LUTs is therefore bounded by $\lceil \log_2(D+1) \rceil + 1$.

Let α denote the number of additions required to perform a modulo $f(x)$ reduction. Then, pipelining the computation of $t(x)$ shortens the critical path if:

$$\lceil \log_2 \alpha \rceil \leq \max(\lceil \log_2 \lceil D/R \rceil \rceil + 1, \lceil \log_2(R+1) \rceil) - 1. \quad (3)$$

Note that this equation does not include any information about routing, and practical results may differ. In our example, $D = 4$ and $R = 2$, thus $\lceil \log_2 \lceil D/R \rceil \rceil + 1 = \lceil \log_2(R+1) \rceil = 2$, and pipelining is therefore of interest only if computation of Equation (2) requires a single addition.

3 Least-Significant Element (LSE) First Algorithms

Least-significant element (LSE) first algorithms mainly consist of two loops. The first one computes the product $p(x)$, while the second one generates successive values of $(x^{iD}b(x)) \bmod f(x)$, $0 \leq i < \lceil m/D \rceil$ (Algorithm 4 and Figure 2a describe the case where $D = 1$). This family requires more hardware resources than MSE first algorithms because of the register, multiplexer, and modulo $f(x)$ reduction involved in the computation of $(x^{iD}b(x)) \bmod f(x)$. Nevertheless, it offers an effective way to compute $(a(x)b(x) + c(x)) \bmod f(x)$: it suffices to initialize register $p(x)$ with $c(x)$ instead of 0 in Algorithm 4. Note that this operation does not involve additional hardware (the control is however a little bit more complex): we can for instance load $c(x)$ in register $q(x)$ and multiply this polynomial by $a_i = 1$. MSE first schemes require an additional shift register to compute $(a(x)b(x)+c(x)) \bmod f(x)$: only D coefficients of $c(x)$ can be added at each step.

Algorithm 4. LSE multiplication over $\mathbb{F}_{p^m}$.

Require: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$.
Ensure: $s(x) = a(x)b(x) \bmod f(x)$
1: $p(x) \leftarrow 0$; $q(x) \leftarrow b(x)$;
2: **for** i in 0 to $m-1$ **do**
3: $\quad p(x) \leftarrow p(x) + a_i q(x)$;
4: $\quad q(x) \leftarrow xq(x) \bmod f(x)$;
5: **end for**

Bertoni *et al.* modified Algorithm 4 so that it processes D coefficients at each iteration [2]. This algorithm can be considered as a transposition of Song and Parhi's proposal to LSE schemes: at each iteration, the sum of D partial products forms a degree-$(m+D-2)$ polynomial $t(x)$ which is added to the intermediate

result $s(x)$. After $\lceil m/D \rceil$ iterations, $s(x)$ is a degree-$(m + D + 2)$ polynomial congruent with $a(x)b(x)$ modulo $f(x)$. A final modulo $f(x)$ reduction is therefore requested to get $p(x) = a(x)b(x) \bmod f(x)$. However, the delay of this operation may sometimes be added to the critical path of an operator connected to the output of such a multiplier. An additional output register avoids this drawback. Another solution consists in reducing $s(x)$ at each step and in performing an additional iteration with $a_j = 0, m \leq j \leq m+D-1$ (Algorithm 5 and Figure 2b). Since $t(x)$ is now equal to zero, $s(x) = (a(x)b(x)) \bmod f(x)$ and it suffices to consider the m least-significant coefficients of $s(x)$ to get the result ($p(x) = s(x) \bmod x^m$).

Algorithm 5. LSE multiplication over $\mathbb{F}_{p^m}$ [2].

Require: A degree-m monic polynomial $f(x) = x^m + f_{m-1}x^{m-1} + \ldots + f_1 x + f_0$ and two degree-$(m-1)$ polynomials $a(x)$ and $b(x)$. We assume that $a_j = 0$, $m \leq j \leq m + D - 1$.

Ensure: $p(x) = a(x)b(x) \bmod f(x)$

1: $p(x) \leftarrow 0$; $q(x) \leftarrow b(x)$;
2: **for** i in 0 to $\lceil m/D \rceil$ **do**
3: $\quad t(x) \leftarrow \sum_{j=0}^{D-1} a_{Di+j} x^j q(x)$;
4: $\quad s(x) \leftarrow s(x) \bmod f(x) + t(x)$;
5: $\quad q(x) \leftarrow (x^D q(x)) \bmod f(x)$;
6: **end for**
7: $p(x) \leftarrow s(x) \bmod x^m$;

Kumar *et al.* further improved the algorithm described in [2] by introducing double accumulator multiplier (DAM) and n-accumulator multiplier (NAM) architectures [9, 6] (Figure 2c). They reduce the critical path by using $R \geq 2$ accumulators to carry out the sum of D partial products. At the end a multiplication, an adder tree combines these intermediate results and a final modulo $f(x)$ operation is performed. An output register should therefore be included in order to avoid adding this combinatorial logic to the critical path of another operator. Another solution consists in pipelining the computation of $t(x)$ (Figure 2d). This operator has the following advantages over DAM and NAM schemes: i) The modulo $f(x)$ reduction is in the accumulation loop. ii) It requires the addition of $(D + 1)$ operands, whereas DAM and NAM architectures compute a sum of $(D + R)$ operands. Thus, a pipelined architecture should lead to smaller circuits than DAM or NAM.

Note that Equation (3) can also be applied to study the efficiency of pipelining. The only difference is that we have to consider the loop computing $x^{iD}b(x) \bmod f(x)$ (Figure 2). If $p = 3$, the multiplexer involves five-input functions. However, several FPGAs embed dedicated multiplexers to efficiently combine two four-input LUTs. The delay of the multiplexer on Figure 2 is therefore equivalent

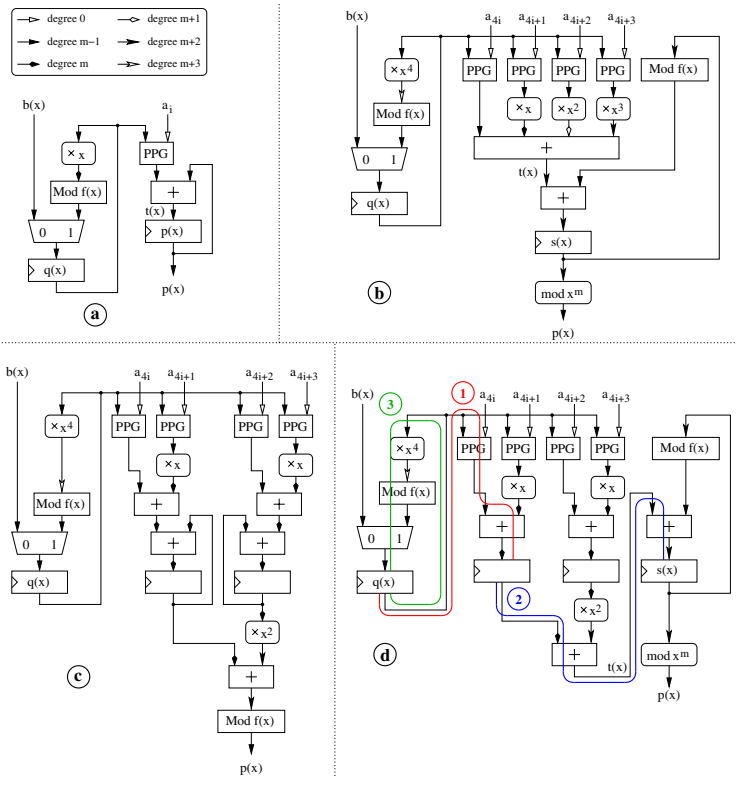

Fig. 2. LSE multiplication over $\mathbb{F}_{p^m}$. (a) Basic algorithm. (b) Modification of the algorithm proposed by Bertoni *et al.* [2] ($D = 4$). (c) Algorithm proposed by Kumar *et al.* [9] ($D = 4$ and $R = 2$). (d) Modification of the algorithm proposed by Kumar *et al.* [9] ($D = 4$ and $R = 2$). Boxes with rounded corners involve only wiring.

to the one of an addition over $\mathbb{F}_3$, and the longest path of this loop includes $\lceil \log_2(D+1) \rceil + 1$ LUTs.

4 Results

In order to easily compare the algorithms described in previous sections, we wrote a VHDL code generator whose inputs are an irreducible polynomial, as well as the desired algorithm and its parameters. Our tool returns a structural VHDL description of the multiplier and some scripts to automatically place-and-route the design and extract its area and critical path. In the following, we discuss results of a series of experiments involving a Spartan-3 XC3S1500 device and ISE WebPACK 8.2.03i.

We considered the following NIST recommended polynomials for studying multiplication over $\mathbb{F}_{2^m}$: $\mathbb{F}_{2^{163}} = \mathbb{F}_2[x]/(x^{163} + x^7 + x^6 + x^3 + 1)$,

Table 1. Multiplication over $\mathbb{F}_{2^m}$ based on the Horner's rule (Algorithm 1)

$\mathbb{F}_{2^m}$	Algorithm	Area [slices]	Delay [ns]	Multiplication time [ns]	Throughput [Mbits/s]	Throughput/ slice [Mbits/s]
$\mathbb{F}_{2^{163}}$	Algorithm 1	85	4.9	798.7	204.1	2.4
	Algorithm 4	179	4.9	798.7	204.1	1.1
$\mathbb{F}_{2^{233}}$	Algorithm 1	122	4.9	1141.7	204.1	1.7
	Algorithm 4	270	5.3	1234.9	188.7	0.7
$\mathbb{F}_{2^{283}}$	Algorithm 1	146	5.3	1499.9	188.7	1.3
	Algorithm 4	322	4.9	1386.7	204.1	0.6
$\mathbb{F}_{2^{409}}$	Algorithm 1	216	6.2	2535.8	161.3	0.7
	Algorithm 4	473	4.9	2004.1	204.1	0.4
$\mathbb{F}_{2^{571}}$	Algorithm 1	311	4.9	2797.9	204.1	0.7
	Algorithm 4	643	4.9	2797.9	204.1	0.3

$\mathbb{F}_{2^{233}} = \mathbb{F}_2[x]/(x^{233} + x^{74} + 1)$, $\mathbb{F}_{2^{283}} = \mathbb{F}_2[x]/(x^{283} + x^{12} + x^7 + x^5 + 1)$, $\mathbb{F}_{2^{409}} = \mathbb{F}_2[x]/(x^{409} + x^{87} + 1)$, and $\mathbb{F}_{2^{571}} = \mathbb{F}_2[x]/(x^{571} + x^{10} + x^5 + x^2 + 1)$. Our first experiment aimed at comparing Horner's rule (Algorithm 1) and the basic LSE first scheme (Algorithm 4). Whereas both algorithms have almost the same critical path, MSE first approach requires less hardware and offers therefore a higher throughput per slice (Table 1).

We then considered several values of D (number of coefficients of $a(x)$ processed at each clock cycle) for Algorithms 2, 3, and 5. Table 2 summarizes the multiplication time measured after placing-and-routing the operators. Recall that Algorithms 2 and 3 respectively require $\lceil m/D \rceil$ and $\lceil m/D \rceil + 1$ iterations. This additional clock cycle explains why Algorithms 2 is often slightly faster. Table 3 reports the throughput per slice of these three algorithms. We conclude from this experiment that MSE first schemes (Algorithms 2 and 3) are almost always more efficient than LSE first algorithm. However, it seems that the choice between these two MSE architectures depends on the irreducible polynomial and parameter D. Note that we obtained slightly different results with an older version of ISE WebPACK. Design tools should also be considered as a parameter when selecting an algorithm. In order to increase the throughput per slice of Algorithm 1, the parameter D should be smaller than 16.

Our next experiment aimed at studying pipelining, DAM, and NAM approaches (Table 4). We selected $D = 32$ and $R = 4$, and generated VHDL descriptions of one MSE first operator (Figure 1d) and two LSE first operators (Figure 2c and 2d). From a theoretical point of view, this choice of parameters should significantly improve the throughput of multiplication over $\mathbb{F}_{2^{283}}$ and $\mathbb{F}_{2^{409}}$: since each modulo $f(x)$ reduction involves a single addition, the multi-operand addition (sum of D partial products) should include the critical path of the circuit. Although experiments carried out for $\mathbb{F}_{2^{233}}$ confirm these assumptions, results obtained for $\mathbb{F}_{2^{409}}$ are somewhat surprising: adding a pipeline stage

Table 2. Multiplication over $\mathbb{F}_{2^m}$ (1). Computation time (in nanoseconds) of Algorithms 2, 3, and 5 on a Spartan-3 XC3S1500 FPGA. Bold numbers highlight the fastest algorithm for each experiment setup.

$\mathbb{F}_{2^m}$	Algorithm	D = 4	D = 8	D = 12	D = 16	D = 20	D = 24	D = 28	D = 32
$\mathbb{F}_{2^{163}}$	Algorithm 2	**203.7**	115.8	79.1	**66.7**	**58.8**	**54.7**	**45.7**	**46.2**
	Algorithm 3	294.4	116.8	**76.2**	72.7	79.1	60.3	51.4	53.6
	Algorithm 5	209.2	**109.7**	89.4	69.3	64.0	58.0	57.4	72.7
$\mathbb{F}_{2^{233}}$	Algorithm 2	**294.4**	**154.2**	119.8	**87.4**	**80.3**	73.8	**93.1**	**95.3**
	Algorithm 3	298.7	155.7	111.7	89.6	81.6	**72.5**	130.4	108.8
	Algorithm 5	345.8	222.4	**111.5**	112.6	95.7	113.6	110.3	104.8
$\mathbb{F}_{2^{283}}$	Algorithm 2	474.6	**201.4**	152.6	138.0	**103.2**	122.0	**121.2**	**94.2**
	Algorithm 3	423.1	303.5	140.7	**119.2**	108.6	**87.6**	123.3	115.3
	Algorithm 5	**363.4**	253.4	**132.1**	161.0	106.8	130.1	131.5	120.9
$\mathbb{F}_{2^{409}}$	Algorithm 2	**518.9**	**272.7**	210.9	**157.8**	201.3	**185.3**	173.4	153.5
	Algorithm 3	535.0	323.7	207.4	177.3	**165.8**	194.8	**168.2**	**150.1**
	Algorithm 5	515.9	348.4	**202.2**	175.6	167.8	196.8	185.0	157.6
$\mathbb{F}_{2^{571}}$	Algorithm 2	**713.9**	444.7	432.9	338.5	**293.9**	267.8	**247.1**	295.8
	Algorithm 3	878.8	**428.1**	**325.9**	**232.9**	347.2	**265.8**	261.8	226.4
	Algorithm 5	779.2	502.4	334.1	307.8	354.4	271.0	251.5	**224.7**

Table 3. Multiplication over $\mathbb{F}_{2^m}$ (2). Throughput/slice [Mbits/s] of Algorithms 2, 3, and 5 on a Spartan-3 XC3S1500 FPGA. Bold numbers indicate a throughput/slice greater than or equal to the one of Horner's rule.

$\mathbb{F}_{2^m}$	Algorithm	D = 4	D = 8	D = 12	D = 16	D = 20	D = 24	D = 28	D = 32
$\mathbb{F}_{2^{163}}$	Algorithm 2	**2.9**	**2.5**	2.0	1.9	1.4	1.2	1.2	1.1
	Algorithm 3	1.9	**2.4**	**2.8**	2.1	1.1	1.2	1.2	1.0
	Algorithm 5	2.1	2.2	2.0	2.0	1.3	1.2	1.0	0.7
$\mathbb{F}_{2^{233}}$	Algorithm 2	**2.1**	**1.9**	**1.7**	**1.7**	1.1	1.0	0.7	0.6
	Algorithm 3	**1.8**	**1.9**	**1.9**	**1.7**	1.1	1.0	0.5	0.5
	Algorithm 5	1.3	1.1	**1.7**	1.3	0.9	0.6	0.5	0.5
$\mathbb{F}_{2^{283}}$	Algorithm 2	**1.3**	**1.4**	1.1	1.0	0.8	0.6	0.5	0.5
	Algorithm 3	**1.3**	**1.0**	**1.5**	**1.3**	0.8	0.8	0.5	0.5
	Algorithm 5	1.2	1.0	**1.4**	0.9	0.8	0.5	0.5	0.4
$\mathbb{F}_{2^{409}}$	Algorithm 2	**1.2**	**1.1**	**1.0**	**0.9**	0.4	0.4	0.4	0.3
	Algorithm 3	**1.0**	**0.9**	**1.0**	**0.8**	0.5	0.4	0.4	0.4
	Algorithm 5	**0.8**	**0.7**	**0.9**	**0.8**	0.5	0.3	0.3	0.3
$\mathbb{F}_{2^{571}}$	Algorithm 2	**0.8**	0.6	0.4	0.4	0.3	0.3	0.2	0.2
	Algorithm 3	0.6	0.6	0.6	0.6	0.3	0.3	0.2	0.2
	Algorithm 5	0.6	0.5	0.5	0.4	0.2	0.3	0.2	0.2

Table 4. Multiplication over $\mathbb{F}_{2^m}$ (3). Comparison of architectures including pipeline stages for $D = 32$ and $R = 4$.

$\mathbb{F}_{2^m}$	Operator	Area [slices]	Delay [ns]	Multiplication time [ns]	Throughput [Mbits/s]	Throughput/ slice [Mbits/s]
$\mathbb{F}_{2^{163}}$	Figure 1d	2158	5.9	47.2	3453.4	1.6
	Figure 2c	2637	6.6	52.8	3087.1	1.2
	Figure 2d	2269	5.9	47.2	3453.4	1.5
$\mathbb{F}_{2^{233}}$	Figure 1d	3458	5.8	58.0	4017.2	1.2
	Figure 2c	3917	7.1	71.0	3281.7	0.8
	Figure 2d	3504	6.2	62.0	3758.1	1.1
$\mathbb{F}_{2^{283}}$	Figure 1d	3649	6.7	73.7	3839.9	1.1
	Figure 2c	4163	7.8	85.8	3298.4	0.8
	Figure 2d	3754	7.7	84.7	3341.2	0.9
$\mathbb{F}_{2^{409}}$	Figure 1d	5406	10.2	153.0	2673.2	0.5
	Figure 2c	6061	9.8	147.0	2782.3	0.5
	Figure 2d	5489	10.6	159.0	2572.3	0.5
$\mathbb{F}_{2^{571}}$	Figure 1d	8329	10.3	206.0	2771.8	0.3
	Figure 2c	10053	10.7	217.0	2631.3	0.3
	Figure 2d	8730	10.1	202.0	2826.7	0.3

does not shorten the critical path. The problem is that our theoretical framework does not include routing delays. This suggest the design of incremental code generators which take advantage of timing information in order to automatically pipeline an arithmetic operator. We also obtained an interesting result with multiplication over $\mathbb{F}_{2^{163}}$. Operators depicted by Figures 1d and 2c have roughly the same throughput as MSE first or LSE first multiplication without pipelining. However, the throughput per slice is improved. It seems that hardware design tools were able to meet timing constraints in both cases. However, adding a pipeline stage made this task easier, thus allowing to build a smaller operator. Table 4 also indicates that pipelining an operator leads to a smaller circuit than DAM or NAM approaches.

We also conducted a series of experiments in $\mathbb{F}_{3^m}$. They involved four pairing-friendly polynomials[2] proposed in [1]: $\mathbb{F}_{3^{97}} = \mathbb{F}_3[x]/(x^{97} + x^{16} - 1)$, $\mathbb{F}_{3^{167}} = \mathbb{F}_3[x]/(x^{167} + x^{92} - 1)$, $\mathbb{F}_{3^{193}} = \mathbb{F}_3[x]/(x^{193} + x^{64} - 1)$, and $\mathbb{F}_{3^{239}} = \mathbb{F}_3[x]/(x^{239} + x^{26} - 1)$. Results are very similar to those obtained in characteristic two. MSE first schemes are almost always more attractive than LSE first algorithms and the choice between Algorithms 2 and 3 depends again on $f(x)$ and D.

5 Conclusion

We studied eight operators performing multiplication over $\mathbb{F}_{p^m}$. Our approach consisted in writing a VHDL code generator in order to investigate a wide

[2] With such polynomials, the cost of computing cube roots over $\mathbb{F}_{3^m}$ is $O(m)$.

solution space. Our results indicate that the choice of an algorithm depends on several parameters, like the irreducible polynomial or the number of coefficients processed at each clock cycle. We plan to improve our tool to automatically pipeline an operator. This task should be based on theoretical results (e.g. Equations 3) and place-and-route information including wire delays. We could for instance generate a VHDL description of a modulo $f(x)$ reduction or an addition tree, place-and-route this operator, and extract timing information, which would then allow one to optimize the architecture of the multiplier.

References

1. P. S. L. M. Barreto. A note on efficient computation of cube roots in characteristic 3. Cryptology ePrint Archive, Report 2004/305, 2004.
2. G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, and T. Wollinger. Efficient GF(p^m) arithmetic architectures for cryptographic applications. In M. Joye, editor, *Topics in Cryptology – CT-RSA 2003*, number 2612 in Lecture Notes in Computer Science, pages 158–175. Springer, 2004.
3. J.-L. Beuchat, M. Shirase, T. Takagi, and E. Okamoto. An algorithm for the η_T pairing calculation in characteristic three and its hardware implementation. Cryptology ePrint Archive, Report 2006/327, 2006.
4. S. E. Erdem, T. Yamk, and Ç. K. Koç. Polynomial basis multiplication over GF(2^m). *Acta Applicandae Mathematicae*, 93(1–3):33–55, September 2006.
5. P. Grabher and D. Page. Hardware acceleration of the Tate Pairing in characteristic three. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 398–411. Springer, 2005.
6. J. Guajardo, T. Güneysu, S. Kumar, C. Paar, and J. Pelzl. Efficient hardware implementation of finite fields with applications to cryptography. *Acta Applicandae Mathematicae*, 93(1–3):75–118, September 2006.
7. T. Kerins, W. P. Marnane, E. M. Popovici, and P.S.L.M. Barreto. Efficient hardware for the Tate Pairing calculation in characteristic three. In J. R. Rao and B. Sunar, editors, *Cryptographic Hardware and Embedded Systems – CHES 2005*, number 3659 in Lecture Notes in Computer Science, pages 412–426. Springer, 2005.
8. T. Kerins, E. Popovici, and W. Marnane. Algorithms and architectures for use in FPGA implementations of identity based encryption schemes. In J. Becker, M. Platzner, and S. Vernalde, editors, *Field-Programmable Logic and Applications*, number 3203 in Lecture Notes in Computer Science, pages 74–83. Springer, 2004.
9. S. Kumar, T. Wollinger, and C. Paar. Optimum digit serial GF(2^m) multipliers for curve-based cryptography. *IEEE Transactions on Computers*, 55(10):1306–1311, October 2006.
10. R. Ronan, C. Ó hÉigeartaigh, C. Murphy, M. Scott, T. Kerins, and W.P. Marnane. An embedded processor for a pairing-based cryptosystem. In *Proceedings of the Third International Conference on Information Technology: New Generations (ITNG'06)*. IEEE Computer Society, 2006.
11. C. Shu, S. Kwon, and K. Gaj. FPGA accelerated Tate pairing based cryptosystem over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006.
12. L. Song and K. K. Parhi. Low energy digit-serial/parallel finite field multipliers. *Journal of VLSI Signal Processing*, 19(2):149–166, July 1998.