Fast Inversion in $GF(2^m)$ with Normal Basis Using Hybrid-Double Multipliers

Reza Azarderakhsh, Kimmo Järvinen, and Vassil Dimitrov

Abstract—Fast inversion in finite fields is crucial for high-performance cryptography and codes. We present techniques to exploit the recently proposed hybrid-double multipliers for fast inversions in binary fields $GF(2^m)$ with normal bases. A hybrid-double multiplier computes a double multiplication, the product of three elements in $GF(2^m)$, with a latency comparable to the latency of single multiplication of two elements. Traditional approaches, such as Itoh-Tsujii, cannot utilize hybrid-double multipliers. We devise a new inversion algorithm based on ternary representations that exploits their potential. The algorithm reduces the latency of inversion significantly for the fields recommended by NIST if hybrid-double multipliers are employed. For example, the algorithm computes an inversion in $GF(2^{163})$ with only five double multiplications whereas the Itoh-Tsujii algorithm requires nine single or double multiplications. We propose a new inverter architecture using this new algorithm and a hybrid-double multiplier. We show that it is faster than the existing techniques by providing ASIC synthesis results using 65-nm CMOS technology. For example, our inverter for $GF(2^{163})$ achieves about 34 percent shorter computation time than an inverter using the Itoh-Tsujii algorithm and a single multiplier.

Index Terms—Finite field, binary extension field, normal basis, inversion, hybrid-double multiplier, Itoh-Tsujii, cryptography, codes, ASIC

1 Introduction

Finite fields $GF(q)$ are commonly used for cryptography and codes. Consequently, efficiency of finite field arithmetic greatly affects the performance of systems using, for example, public-key cryptography. For this reason, efficient implementation of arithmetic operations in $GF(q)$ has been studied extensively. Binary extension fields—that is, fields $GF(q)$ for which $q = 2^m$—are of particular interest, especially, for hardware implementations because they allow carry-less arithmetic and, thus, faster and smaller implementations.

The most important operations in binary fields are additions, multiplications, squarings (an element multiplied by itself), and inversions. We focus on inversions which are computationally more demanding than the other field operations. There are basically two ways to find the multiplicative inverse of an element $A \in GF(2^m)$—that is, to find $A^{-1} \in GF(2^m)$ such that $A \times A^{-1} = 1$: extended euclidean algorithm (EEA) and Fermat's little theorem (FLT). The latter is based on the exponentiation A^{2^m-2} in $GF(2^m)$ which can be computed by a series of multiplications and squarings. Standard exponentiation methods lead to $O(m)$ multiplications [1] but the method introduced by Itoh and Tsujii in [2] exponentiations, reduces the complexity to $O(\log m)$. FLT is often more suitable for hardware implementations because it can be realized by reusing a multiplier (and a squarer) whereas EEA typically requires more additional logic.

Binary fields with the so-called normal basis are particularly suitable for hardware implementations because squarings can be carried out by a simple cyclic shift of the bit vector representing an element. Let $A, B, C, D, E \in GF(2^m)$. A single multiplication is the following 'normal' multiplication with two operands:

$$C = A \times B. \tag{1}$$

There have been many proposals for multiplier architectures for $GF(2^m)$ with normal basis including [1], [3], [4], [5], [6]. A combination of two consecutive single multiplications $C = A \times B$ and $E = C \times D$ gives the following multiplication involving three operands:

$$E = A \times B \times D. \tag{2}$$

We call it a double multiplication. Respectively, a component computing (2) is called a double multiplier. Recently, Azarderakhsh and Reyhani-Masoleh [6] presented the first double multiplier architecture that can compute (2) with essentially the same latency as (1). They achieved this by combining and interleaving a digit-level parallel-in serial-out (DL-PISO) and serial-in parallel-out (DL-SIPO) multipliers. Because of this hybrid nature, they called it the hybrid-double multiplier.

We explore possibilities to exploit hybrid-double multipliers for computing inversions using FLT. The exponentiations used in inversions over $GF(2^m)$ (including the Itoh-Tsujii (IT) algorithm [2]) are notoriously difficult to parallelize and neither do they allow efficient utilization of double multiplications. We propose a variation of the IT algorithm that is based on ternary representations rather than on binary representations of the regular IT. We show that this variation is particularly well suited for double multiplications and that it provides significant reductions in latency when compared to existing techniques if double multipliers are available. We back up our claims with application specific integrated circuit (ASIC) results using 65-nm CMOS technology.

2 $GF(2^m)$ with Gaussian Normal Basis

Binary fields $GF(2^m)$ can be constructed by using a normal basis $N = \{\beta, \beta^2, \beta^{2^2}, \ldots, \beta^{2^{m-1}}\}$ where $\beta \in GF(2^m)$ is a normal element of $GF(2^m)$, that is, $\beta, \beta^2, \ldots, \beta^{2^{m-1}}$ are linearly independent. Then, any element $A = (a_0, a_1, \ldots, a_{m-1}) \in GF(2^m)$ can be represented as [7]

$$A = \sum_{i=0}^{m-1} a_i \beta^{2^i}, \quad \text{where } a_i \in GF(2). \tag{3}$$

Gaussian normal basis (GNB) [8] is a special class of normal basis which is included in the IEEE 1363 [9] and NIST [10] standards for ECDSA and exists for every $m > 1$ that is not divisible by 8 [11].

Definition 1 [11]. *Let m and T be positive integers such that $p = mT + 1$ is a prime number and $\gcd(mT/k, m) = 1$, where k is the multiplicative order of 2 modulo p. Let α be a primitive $(mT + 1)$th root of unity in $GF(2^{Tm})$. Then, for any primitive Tth root of unity $\tau \in Z_p$, $\beta = \sum_{i=0}^{T-1} \alpha^{\tau^i}$ generates a normal basis of $GF(2^m)$ over $GF(2)$ as $N = \{\beta, \beta^2, \ldots, \beta^{2^{m-1}}\}$ which is called the Gaussian normal basis of type T.*

For the five binary fields recommended by NIST, that is, $m = 163, 233, 283, 409,$ and 571, the values of T are $4, 2, 6, 4,$ and 10, respectively.

2.1 Addition and Squaring

Let A and B be elements in $GF(2^m)$. Then, addition $A + B$ is given by $\sum_{i=0}^{m-1} a_i \beta^{2^i} + \sum_{i=0}^{m-1} b_i \beta^{2^i} = \sum_{i=0}^{m-1} (a_i + b_i) \beta^{2^i}$ where $a_i + b_i$ is computed in $GF(2)$ and is a logical exclusive-or (xor). Hence, addition in $GF(2^m)$ is a bitwise xor of the bit vectors representing A and B.

- *R. Azarderakhsh is with the Department of Computer Engineering, Rochester Institute of Technology, Rochester, NY 14623, USA. E-mail: rxaeec@rit.edu.*
- *K. Järvinen is with the Department of Information and Computer Science, Aalto University, Konemiehentie 2, FI-02150 Espoo, Finland. E-mail: kimmo.jarvinen@aalto.fi.*
- *V. Dimitrov is with the Department of Electrical and Computer Engineering, University of Calgary, 2500 University Dr. NW, Calgary, AB T2N 1N4, Canada. E-mail: vdimitro@ucalgary.ca.*

Manuscript received 14 Jun. 2012; revised 10 Oct. 2012; accepted 14 Oct. 2012; date of publication 24 Oct. 2012; date of current version 5 Mar. 2014. Recommended for acceptance by J. Villalba. For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below. Digital Object Identifier no. 10.1109/TC.2012.265

TABLE 1
The Upper Bounds of the Area and Time Complexities of Digit-Level Type T GNB Multipliers over $GF(2^m)$ [6]

Multiplier Architecture	# AND gates	# XOR[1] gates	# Reg.	Critical-Path delay	Output
DL-PIPO	dm	$\leq v_p(T-1)+dm$	$3m$	$T_A+(\lceil\log_2 T\rceil+\lceil\log_2(d+1)\rceil)\,T_X$	Parallel
DL-SIPO	dm	$\leq v_s(T-1)+dm$	$2m$	$T_A+(\lceil\log_2 T\rceil+\lceil\log_2(d+1)\rceil)\,T_X$	Parallel
DL-PISO	dm	$\leq v_s(T-1)+d(m-1)$	$2m$	$T_A+(\lceil\log_2 T\rceil+\lceil\log_2 m\rceil)\,T_X$	Serial

1. $v_p=\frac{d(m-1)}{2}$ and $v_s=d(m-1)-\frac{d(d-1)}{2}$.

Squaring is also very simple. Squaring (3) gives $A^2=\sum_{i=0}^{m-1} a_i\beta^{2^{i+1}}$ and because $\beta^{2^m}=\beta$ based on FLT, squarings are cyclic right shifts: $A^{2^e}=(a_{m-e},\ldots,a_{m-1},a_0,a_1,\ldots,a_{m-e-1})=A\gg e$.

2.2 Multiplication

Let A and B be as above and assume their product is $C=(c_0,c_1,\ldots,c_{m-1})=A\times B$. Then, c_0 can be obtained as follows [9], [12]:

$$c_0=a_0b_1+\sum_{i=1}^{m-1}a_i\left(\sum_{j=1}^{T}b_{R(i,j)}\right), \tag{4}$$

where $R(i,j)\in[0,m-1]$ are elements of an $m-1\times T$ multiplication matrix $\mathbf{R}$. To obtain c_l, one adds "$l \bmod m$" to all indices in (4).

Single digit-level GNB multipliers can be implemented based on (4) in three different architectures including digit-level parallel-in parallel-out (DL-PIPO), digit-level serial-in parallel-out , and digit-level parallel-in serial-out. All these multipliers require $q=\lceil\frac{m}{d}\rceil$, where $1\leq q\leq m$ and $1\leq d\leq m$, clock cycles to generate all m coordinates of $C=A\times B$. As shown in the comparison presented in Table 1 [6], the DL-PIPO architecture has the lowest time and area complexities and, therefore, we employ it as a single multiplier for inversions using the IT algorithm.

In [6], a new digit-level hybrid-double multiplier was proposed that performs double multiplications with a latency of $\lceil\frac{m}{d}\rceil+1$ clock cycles assuming that one clock cycle is required to load the output of the first multiplier to the input of the second multiplier. The structure of the hybrid-double multiplier is illustrated in Fig. 1. The registers $\langle X\rangle$, $\langle Y\rangle$, and $\langle F\rangle$ should be preloaded[1] with the operands A, B, and D, respectively, and the register $\langle Z\rangle$ should be cleared to $0\in GF(2^m)$. The result E is available in parallel after $M=\lceil\frac{m}{d}\rceil+1$ clock cycles in the register $\langle Z\rangle$. The area and time complexities of the hybrid-double multiplier are as follows.

Proposition 2 ([6]). The hybrid-double multiplier architecture requires $\leq 2v_s(T-1)+2dm-d$, $v_s=d(m-1)-d(d-1)/2$ XOR gates, $2dm$ AND gates, four m-bit registers and one d-bit register. Its critical-path delay is $T_A+(\lceil\log_2 T\rceil+\lceil\log_2 m\rceil)T_X$, where T_A and T_X are the delay of an AND gate and XOR gate, respectively.

The multiplication delay is $T_M=t_{CP}\times M$, where t_{CP} is the critical-path delay of the multiplier [6].

2.3 Inversion

Based on FLT, we know that $A^{2^m-1}=1$ for all $A\neq 0\in GF(2^m)$ which gives

$$A^{-1}=A^{2^m-2}. \tag{5}$$

Applying traditional exponentiation techniques to (5) gives the following:

$$\begin{aligned}A^{-1}&=A^{2^m-2}=A^{2(1+2+\cdots+2^{m-2})}\\&=B^{1+2+\cdots+2^{m-2}}=B\times B^2\times\cdots\times B^{2^{m-2}},\end{aligned} \tag{6}$$

where $B=A^2$. Clearly, this straightforward computation requires $m-2$ multiplications and $m-1$ squarings [1].

Each $+$-sign in the exponent of B results in a multiplication and each power of 2 yields squarings. Hence, the goal is to find a decomposition for $1+2+2^2+\cdots+2^{m-2}$ where the number of additions is as small as possible by using only $+$'s, powers of 2, and parentheses.

In 1988, Itoh and Tsujii [2] introduced an algorithm which finds decompositions with very few $+$'s. Consequently, it yields dramatic reductions in the number of multiplications needed in the exponentiation—from $O(m)$ to $O(\log m)$. The IT algorithm is based on the observation that $1+2+2^2+\cdots+2^{m-2}$ can be decomposed as shown in (7). The IT algorithm requires exactly $\lfloor\log_2(m-1)\rfloor+H_2(m-1)-1$ multiplications, where $H_2(m-1)$ is the Hamming weight (the number of ones in the binary expansion of $m-1$), and $m-1$ squarings [2].

$$\begin{aligned}&1+2^n+2^{2n}+\cdots+2^{(k-2)n}\\&\quad=\begin{cases}(1+2^n)\times(1+2^{2n}+2^{4n}+\cdots+2^{(k-3)n}),\\ \qquad\qquad\text{if } k-1\equiv 0 \pmod 2\\ 1+2^n\times(1+2^n)\times(1+2^{2n}+2^{4n}+\cdots+2^{(k-4)n}),\\ \qquad\qquad\text{if } k-1\equiv 1 \pmod 2,\end{cases}\end{aligned} \tag{7}$$

For example, for the NIST field $GF(2^{163})$, the IT algorithm operates as follows. First, 162 is even so we use the first branch of (7) and get $1+2+\cdots+2^{161}=(1+2)\times(1+2^2+\cdots+2^{160})$. Next, $160=2\times 80$ and 81 is odd so we take the second branch and get $1+2^2+2^{2\times 2}+\cdots+2^{2\times 80}=1+2^2\times(1+2^2)\times(1+2^{2\times 2}+\cdots+2^{2\times 78})$. When we continue like this, we have the following decomposition:

$$\begin{aligned}1+2+2^2+\cdots+2^{161}=(1+2)\times(1+2^2\times\cdots\\(1+2^2)\times(1+2^4)\times(1+2^8)\times(1+2^{16})\times\cdots\\(1+2^{32}\times(1+2^{32})\times(1+2^{64})))\ .\end{aligned} \tag{8}$$

This results in Algorithm 1.

Algorithm 1: Inversion in $GF(2^{163})$ with the IT algorithm [2]

Input: $A\in GF(2^{163}), A\neq 0$
Output: $B=A^{-1}$
1 $B\leftarrow A\gg 1$
2 $B,C\leftarrow B\times(B\gg 1)$
3 $B\leftarrow B\times(B\gg 2)$
4 $B\leftarrow B\times(B\gg 4)$
5 $B\leftarrow B\times(B\gg 8)$
6 $B,D\leftarrow B\times(B\gg 16)$
7 $B\leftarrow B\times(B\gg 32)$
8 $B\leftarrow B\times(B\gg 64)$
9 $B\leftarrow D\times(B\gg 32)$
10 $B\leftarrow C\times(B\gg 2)$
11 **return** $B=A^{-1}$

1. Because of the one clock cycle delay, $\langle F\rangle$ can be preloaded one clock cycle later than $\langle X\rangle$ and $\langle Y\rangle$.

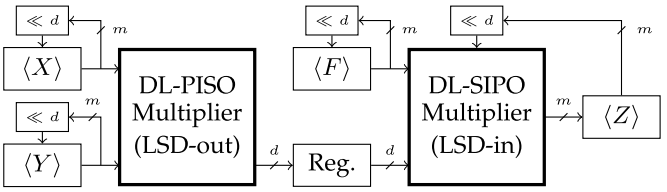

Fig. 1. Architecture of the hybrid-double multiplier [6].

The IT algorithm computes the inverses over $GF(2^m)$ with very low number of multiplications—in some cases, the lowest possible number. Nevertheless, it has been realized that sometimes one can save even more multiplications by using special decompositions of $m-1$. The algorithm proposed by Takagi, Yoshiki, and Takagi (TYT) [13] is based on exploring such ideas. Unfortunately, for the NIST fields it does not lead to savings.

The IT algorithm is based on a specific decomposition of $m-1$. This decomposition leads to computational algorithm that is purely sequential. The previous attempts [14] to parallelize inversions have relied on using parallel computation of squarings and square roots. No attempts have been made in the open literature in finding suitable decompositions of $m-1$ that can lead to efficient parallel inversion architectures.

3 The IT Algorithm and Double Multiplications

Decompositions given by the IT algorithm are particularly unsuitable for double multiplications. As shown in (7), they consist of terms of the form $(1+2^n)$ and $1+2^n\times(1+2^n)$.

The $(1+2^n)$ terms result in $B\times(B\gg n)$, which do not take use of the third operand in a double multiplication; that is, we would compute $B\times(B\gg n)\times 1$. Two consecutive terms of this form yield the computation

$$\left(B^{1+2^{n_1}}\right)^{1+2^{n_2}}=\left(B\times B^{2^{n_1}}\right)^{1+2^{n_2}}=C\times C^{2^{n_2}}, \tag{9}$$

where $C=B\times B^{2^{n_1}}$. This cannot be combined to one double multiplication any better because the result of the first is needed for both operands of the second. Consequently, the cost of taking the even branch of (7) is one double multiplication.

Commonly the $1+2^n\times(1+2^n)$ terms have a decomposition attached to the right and we have $1+2^n\times(1+2^n)\times(\cdots)$. In this case, we cannot apply double multiplications any better than above because the multiplications of the $2^n\times(1+2^n)\times(\cdots)$ terms must be calculated before the first multiplication can be computed. The cost of taking the odd branch of (7) is two double multiplications with the following exception.

If $1+2^n\times(1+2^n)$ is the last (the rightmost) term in a decomposition, then we get $1+2^n\times(1+2^n)=1+2^n+2^{2n}$. This term is ideal for double multiplications as it can be computed as $B\times(B\gg n)\times(B\gg 2n)$ and, consequently, it costs only one double multiplication. The last term has this form if and only if the second highest bit is one in the binary expansion of $m-1$; we denote this bit by b.

We conclude that the number of double multiplications with the IT algorithm is $\lfloor\log_2(m-1)\rfloor+H_2(m-1)-b-1$. Hence, if the latencies of single and double multiplications are approximately the same, one saves the cost of one multiplication if and only if $b=1$; otherwise, the cost is the same. Given the fact that the area of a hybrid-double multiplier is approximately twice the area of a single multiplier [6], it is not practical to use hybrid-double multipliers with the IT algorithm.

4 Ternary Itoh-Tsujii Algorithm

As shown in Section 3, decompositions with many terms of the form $1+2^{n_1}+2^{n_2}$ would be suitable for double multiplications. Hence, the task is to maximize their number in a decomposition of $1+2+2^2+\cdots+2^{m-2}$. The following algorithm creates them on every iteration.

In the IT algorithm, we could use the factorization $(1+2^n)\times(1+2^{2n}+2^{4n}+\cdots+2^{(k-3)n})$ if $k-1$ was even. Similarly, we can use the factorization $(1+2^n+2^{2n})\times(1+2^{3n}+2^{6n}+\cdots+2^{(k-4)n})$ if and only if $k-1$ is divisible by 3. If $k-1$ is not divisible by 3 but $k-2$ is ($k-1\equiv 1 \pmod 3$), then we can do the same for $2^n+2^{2n}+\cdots+2^{(k-2)n}$ and we get $1+2^n+2^{2n}+\cdots+2^{(k-2)n}=1+2\times(1+2^n+2^{2n})\times(1+2^{3n}+2^{6n}+\cdots+2^{(k-5)n})$. The case, where $k-1\equiv 2 \pmod 3$, can be dealt similarly and we can devise a Ternary Itoh-Tsujii algorithm based on (10).

$$1+2^n+\cdots+2^{(k-2)n}=\begin{cases}(1+2^n+2^{2n})\times(1+2^{3n}+2^{6n}+\cdots+2^{(k-4)n}), & \text{if } k-1\equiv 0 \pmod 3\\ 1+2^n\times(1+2^n+2^{2n})\times(1+2^{3n}+2^{6n}+\cdots+2^{(k-5)n}), & \text{if } k-1\equiv 1 \pmod 3\\ 1+2^n+2^{2n}\times(1+2^n+2^{2n})\times(1+2^{3n}+2^{6n}+\cdots+2^{(k-6)n}), & \text{if } k-1\equiv 2 \pmod 3.\end{cases} \tag{10}$$

Clearly, the first branch of (10) requires only one double multiplication whereas the second and the third branches both require two. Hence, the exact number of double multiplications is $\lfloor\log_3(m-1)\rfloor+H_3(m-1)+c-1$, where $H_3(m-1)$ is the ternary Hamming weight, the number of nonzeros in the ternary expansion of $m-1$, and $c=0$ if the most significant ternary digit is one, otherwise $c=1$. On average, the TIT algorithm requires $\frac{5}{3}\log_3(m-1)-0.5\approx 1.05\log_2(m-1)-0.5$ double multiplications whereas the IT algorithm requires $1.5\log_2(m-1)-1.5$. We can therefore expect an average decrease of about 30 percent in latency. Hence, the decompositions of the TIT algorithm are, indeed, very suitable for double multiplications.

Consider the NIST field $GF(2^{163})$, for example. The IT algorithm gave the decomposition of (8). Because $m-1=162=\langle 10100010\rangle_2$ and $b=0$, the number of both single and double multiplications is 9. The number $162=\langle 20000\rangle_3$ is particularly 'nice' for the TIT algorithm which returns: $(1+2+2^2)\times(1+2^3+2^6)\times(1+2^9+2^{18})\times(1+2^{27}+2^{54})\times(1+2^{81})$. This can be computed with only five double multiplications as shown in Algorithm 2. Therefore, latency decreases by 44.4 percent.

Algorithm 2: Inversion in $GF(2^{163})$ with the TIT algorithm and double multiplications

Input: $A\in GF(2^{163}), A\neq 0$
Output: $B=A^{-1}$
1 $B\leftarrow A\gg 1$
2 $B\leftarrow B\times(B\gg 1)\times(B\gg 2)$
3 $B\leftarrow B\times(B\gg 3)\times(B\gg 6)$
4 $B\leftarrow B\times(B\gg 9)\times(B\gg 18)$
5 $B\leftarrow B\times(B\gg 27)\times(B\gg 54)$
6 $B\leftarrow B\times(B\gg 81)\times 1$
7 **return** $B=A^{-1}$

In order to provide experimental evidence, we calculated decompositions with the IT and TIT algorithms for $GF(2^m)$, $3\le m\le 1,023$. The results are shown in Fig. 2. In this interval, the TIT algorithm requires fewer double multiplications than the IT algorithm in 958 cases, as many in 60 cases, and more in 3 cases. On average, the TIT algorithm saves 3.1146 double multiplications. The average decrease in the number of double multiplications is

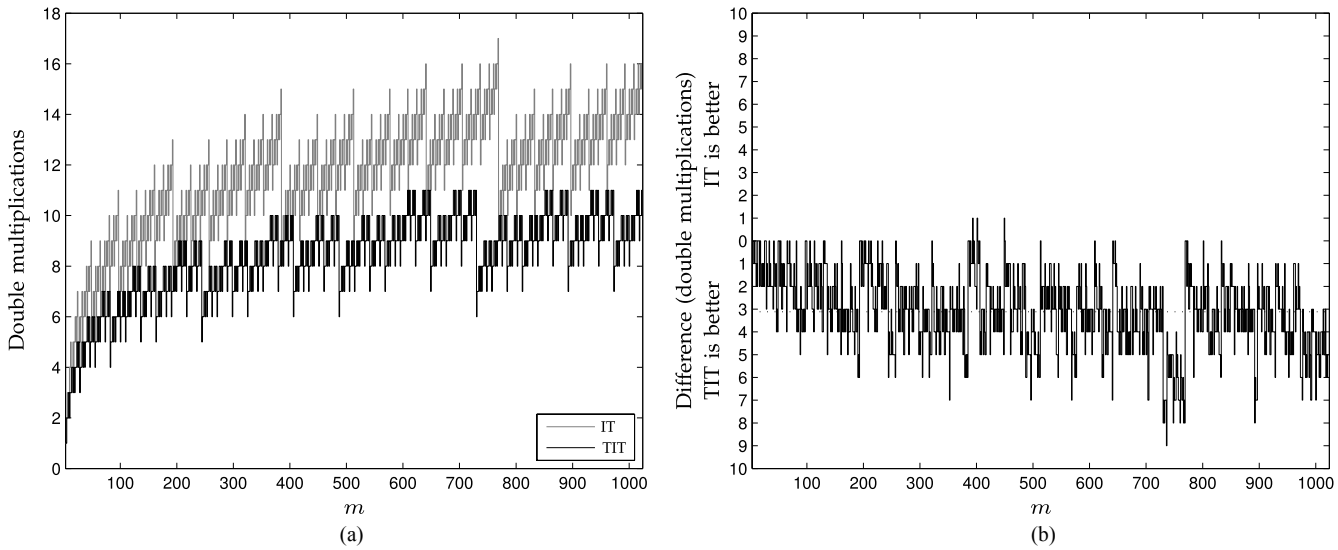

Fig. 2. Comparison of the Itoh-Tsujii and Ternary Itoh-Tsujii algorithms for $GF(2^m)$, $3 \leq m \leq 1{,}023$. (a) shows the numbers of double multiplications for the two algorithms and (b) shows the differences of these numbers.

27.0 percent which is in line with the theoretical calculations above. The maximum savings are nine double multiplications for $GF(2^{736})$ (7 versus 16) and -56.3 percent for $GF(2^{730})$ (6 versus 14). Indeed, the IT algorithm outperforms the TIT algorithm by one double multiplication (+11.1 percent, 10 versus 9) in three cases: $GF(2^{393})$, $GF(2^{401})$, and $GF(2^{449})$. For these fields, $m-1$ has a particularly 'bad' ternary expansion and a 'nice' binary expansion; for example, $392 = \langle 110001000 \rangle_2 = \langle 112112 \rangle_3$. Because the fields recommended by NIST in [10] have significant practical relevance, we show the decompositions returned by the IT and TIT algorithm and their costs in Table 2. The TIT algorithm provides improvements for all NIST fields except $GF(2^{233})$; the best case is the above demonstrated $GF(2^{163})$.

5 Inverter Architectures

5.1 IT Inverter with a Single Multiplier

The IT algorithm comprises multiplications and exponentiations to powers of two. Hence, we need a multiplier, a squarer, and a register file for temporary variables in order to implement an inverter. Let $\mathcal{E} = \{e_1, e_2, \ldots, e_C\}$ be the set of unique powers of two needed in an inversion. As we are aiming to fast inversion, we always perform all exponentiations to powers of two that we can at once with a C-to-1 multiplexer selecting one of the rotated inputs $B^{2^{e_i}} = B \gg e_i$ with $e_i \in \mathcal{E}$. For example, Algorithm 1 gives $\mathcal{E}_{\mathrm{IT},163} = \{1, 2, 4, 8, 16, 32, 64\}$ and we need a 163-bit 7-to-1 multiplexer.

The inverter architecture using the IT algorithm and a single DL-PIPO multiplier is presented in Fig. 3. Similar architectures (with small differences) have been used, for example, in [15], [16], [17].

For example, Algorithm 1 operates as follows. The rotation attached to the input A gives $B \leftarrow A \gg 1$. OP1 is loaded with B by selecting $s_0 = 1$ and OP2 is loaded with $B \gg 1$ by selecting $s_1 = 0$ and $s_2 = e_1 = 1$. The result of this multiplication is stored in C in the 2×163-bit register file. Next, the result of the multiplication is fed to OP1 ($s_0 = 0$) and rotated by two before feeding it to OP2 ($s_1 = 1, s_2 = e_2 = 2$). We continue like this up to line 8. Then, we fetch D from the register file and load it in OP1 with $s_0 = 2$. The result of the multiplication is rotated by 32 by selecting $s_1 = 1$ and $s_2 = e_6 = 32$. The line 10 is handled similarly and, after that, the result is ready.

5.2 TIT Inverter with a Hybrid-Double Multiplier

The inverter architecture for the TIT algorithm with a hybrid-double multiplier is close to the architecture of Fig. 3 with a few differences, the most important being the hybrid-double multiplier with three operands: OP1, OP2, and OP3. The architecture is presented in Fig. 4.

The architecture must also be able to exponentiate operands so that when the value for OP2 is exponentiated to the 2^{e_1}th power, the value for OP3 is exponentiated to the 2^{2e_i}th power. This could be done by adding a multiplexer with exponents $2e_1, 2e_2, \ldots, 2e_C$ for OP3. However, we save some area by reusing the multiplexer of OP2 also for OP3. Because the DL-SIPO

TABLE 2
Decompositions Given by the IT and TIT Algorithms for the NIST Fields

m	Algorithm, decomposition, number of double multiplications, reduction %		
163	IT: $(1+2)(1+2^2(1+2^2)(1+2^4)(1+2^8)(1+2^{16})(1+2^{32}(1+2^{32})(1+2^{64})))$	9	
	TIT: $(1+2+2^2)(1+2^3+2^6)(1+2^9+2^{18})(1+2^{27}+2^{54})(1+2^{81})$	5	**−44.4**
233	IT : $(1+2)(1+2^2)(1+2^4)(1+2^8(1+2^8)(1+2^{16})(1+2^{32}(1+2^{32})(1+2^{64}(1+2^{64}))))$	9	
	TIT: $(1+2(1+2+2^2)(1+2^3+2^6(1+2^3+2^6)(1+2^9(1+2^9+2^{18})(1+2^{27}+2^{54}(1+2^{27}+2^{54})(1+2^{81})))))$	9	**0.0**
283	IT: $(1+2)(1+2^2(1+2^2)(1+2^4)(1+2^8(1+2^8)(1+2^{16}(1+2^{16})(1+2^{32})(1+2^{64})(1+2^{128}))))$	11	
	TIT: $(1+2+2^2)(1+2^3(1+2^3+2^6)(1+2^9(1+2^9+2^{18})(1+2^{27}(1+2^{27}+2^{54})(1+2^{81}+2^{162}))))$	8	**−27.3**
409	IT: $(1+2)(1+2^2)(1+2^4)(1+2^8(1+2^8)(1+2^{16}(1+2^{16})(1+2^{32})(1+2^{64})(1+2^{128}(1+2^{128}))))$	10	
	TIT: $(1+2+2^2)(1+2^3(1+2^3+2^6)(1+2^9+2^{18})(1+2^{27}+2^{54})(1+2^{81}+2^{162}(1+2^{81}+2^{162})))$	7	**−30.0**
571	IT: $(1+2)(1+2^2(1+2^2)(1+2^4)(1+2^8(1+2^8)(1+2^{16}(1+2^{16})(1+2^{32}(1+2^{32})(1+2^{64})(1+2^{128})(1+2^{256})))))$	13	
	TIT: $(1+2+2^2)(1+2^3(1+2^3+2^6)(1+2^9+2^{18})(1+2^{27}+2^{54})(1+2^{81}(1+2^{81}+2^{162})(1+2^{243})))$	8	**−38.5**

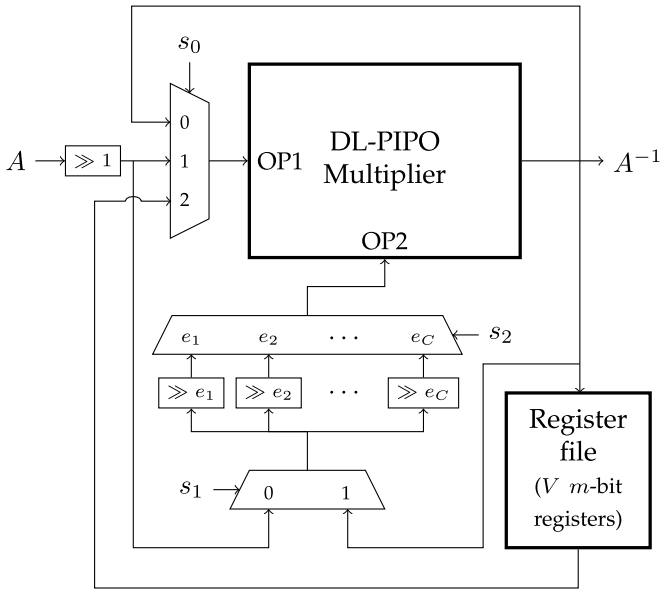

Fig. 3. The inverter architecture using a single multiplier.

multiplier of the hybrid-double multiplier starts computing its part one clock cycle later than the DL-PISO multiplier (due to the d-bit register between two multipliers) [6], we have one clock cycle to compute the second exponentiation from the result of the first (in OP2). This is done by rotating the value from OP2 through the same multiplexers ($s_1 = 0$ and $s_2 = e_i$) again, which results in $(B \gg e_i) \gg e_i = B \gg 2e_i$.

We demonstrate how the architecture operates for the NIST field $GF(2^{409})$; the algorithm is given in Algorithm 3. This example was selected because the algorithm includes all types of terms that can occur in TIT.

The rotation attached to the input A gives $B \leftarrow A \gg 1$ and we directly proceed to line 2. OP1 is loaded with B by selecting $s_0 = 1$ and OP2 is loaded with $B \gg 1$ by selecting $s_1 = 1$ and $s_2 = e_1 = 1$. We compute $B \gg 2$ by taking the value of OP2—that is, $B \gg 1$—and exponentiating it again by 2^{e_1} resulting $(B \gg 1) \gg 1 = B \gg 2$. We do this by selecting $s_1 = 1$ and $s_2 = e_1 = 1$ and we store this value in OP3 by selecting $s_3 = 0$. The result of the multiplication is stored in C in the 2×409-bit register file. We continue similarly for lines 3-6 except that, now, we always select the result of the previous multiplication instead of $A \gg 1$ by setting $s_0 = 0$, $s_1 = 3$, and $s_2 = e_i$. The multiplication on line 7 is handled differently because we use a value (D) from the register file. First, we fetch D and load it in OP1 by selecting $s_0 = 2$. Second, at the same time, the result of the previous multiplication is rotated using the multiplexers ($s_1 = 3$, $s_2 = e_6 = 162$) and stored in OP2. Third, in the next clock cycle, we fetch D again, rotate it ($s_1 = 2$, $s_2 = e_5 = 81$) and store it in OP3 ($s_3 = 0$). The last multiplication on line 8 takes use of only two operands and we set OP3 to one by selecting $s_3 = 1$. After this multiplication is completed, the inverse of A is available in the output. Now, we had $\mathcal{E} = \{1, 3, 9, 27, 81, 162\}$. We could compute $B \gg 162$ on line 7 as $(B \gg 81) \gg 81$ and take $e_6 = 162$ out of $\mathcal{E}$. Because this would slightly increase the complexity of control and the latency of inversion, we preferred to implement the inverter as shown above.

The architecture gets simpler for certain m. If $H_3(m-1) = 1$, then $V = 0$, $s_0 \in \{0, 1\}$, and $s_1 \in \{0, 1, 3\}$. The NIST field $GF(2^{163})$ is such an example. Similarly, $V = 0$ for the IT architecture if $H_2(m-1) = 1$, that is, $m = 2^n + 1$. If the $k - 1 \equiv 2 \pmod 3$ branch of (10) is never taken, then values from the register file are never needed for OP2 (or OP3) and, consequently, $s_1 \in \{0, 1, 3\}$. If $m - 1 = 3^n$, then all multiplications have three operands and the multiplexer for OP3 is not needed.

Algorithm 3: Inversion in $GF(2^{409})$ with the TIT algorithm and double multiplications

Input: $A \in GF(2^{409}), A \neq 0$
Output: $B = A^{-1}$
1 $B \leftarrow A \gg 1$
2 $B, C \leftarrow B \times (B \gg 1) \times (B \gg 2)$
3 $B \leftarrow B \times (B \gg 3) \times (B \gg 6)$
4 $B \leftarrow B \times (B \gg 9) \times (B \gg 18)$
5 $B, D \leftarrow B \times (B \gg 27) \times (B \gg 54)$
6 $B \leftarrow B \times (B \gg 81) \times (B \gg 162)$
7 $B \leftarrow D \times (D \gg 81) \times (B \gg 162)$
8 $B \leftarrow C \times (B \gg 3) \times 1$
9 **return** $B = A^{-1}$

5.3 IT Inverter with a Hybrid-Double Multiplier

Although it was concluded in Section 3 that double multipliers are not feasible for the IT algorithm, we still designed inverters for this case in order to receive experimental evidence. The architecture of these inverters is the same given in Fig. 4 with the exception that $s_1 \in \{0, 1, 3\}$ because values from the register file are never needed for OP2 or OP3. These inverters have a shorter latency than the inverter of Fig. 3 only if $b = 1$.

6 ASIC Implementation Results

In this section, we evaluate the area and time requirements of the inverters from Section 5 on a 65-nm CMOS ASIC. We used Synopsys Design Vision, which is a GUI for Synopsys Design Compiler tools [18], and VHDL for modeling the architectures. For the ASIC implementations, the map effort was set to medium with a target clock period of 5 ns. We report the area (μm^2) and timing (ns) for different field and digit sizes.

We implemented inverters for all NIST fields [10] except $GF(2^{233})$ which was unsuitable for double multiplications as explained in Section 4. The decompositions used in the implementations can be found in Table 2. We chose digit sizes so that they reduce the latency while increasing d. The implementation results are reported in Table 3. The total time of inversion is calculated by multiplying the latency (number of clock cycles) by the critical-path delay. The latencies of single and double multiplications are $q = \lceil m/d \rceil$ and $q + 1$, respectively.

Inversion using the TIT algorithm with a hybrid-double multiplier is faster than the IT schemes for all the reported field sizes. For example, the fastest inversion using the IT algorithm over $GF(2^{163})$ is computed in 125.5 ns with $d = 41$ but the TIT algorithm with a hybrid-double multiplier needs only 83.2 ns with $d = 33$; that is, it is about 34 percent faster. Our results also verify that the IT algorithm with a hybrid-double multiplier is not efficient. In fact, the worse critical-path delays caused by the increased design complexity always make it slower than using a single multiplier.

It is worth mentioning that the faster inversion times were achieved at the expense of larger area requirements. However, we stress that such performance is not possible with the IT algorithm due to the data dependencies. The proposed scheme is applicable primarily for high performance cryptographic applications where the aim is to maximize the speed of inversions.

However, the new scheme improves also the area-time efficiency for certain fields. The NIST field $GF(2^{163})$ is an example of such a field as shown in Fig. 5. For applications requiring tradeoffs between time and area, the digit-size should be chosen so that

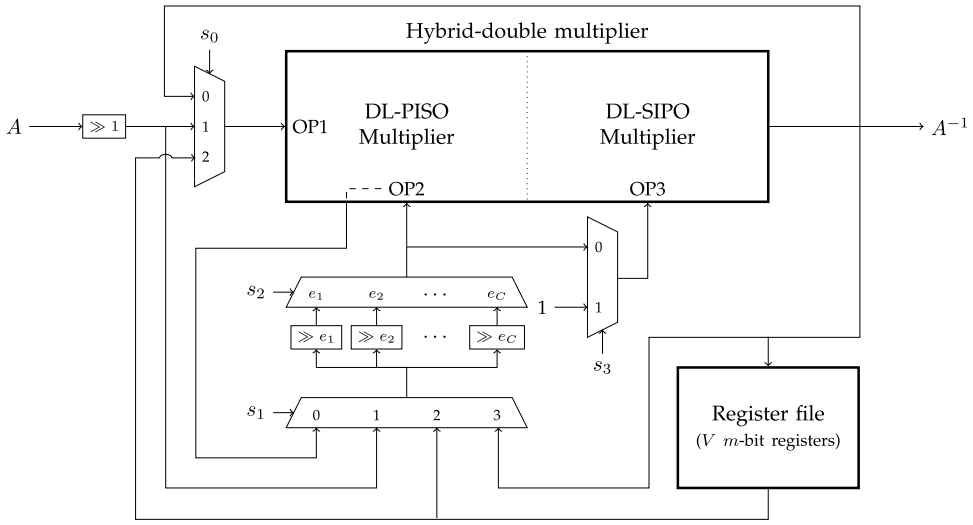

Fig. 4. The proposed inverter architecture using a hybrid-double multiplier.

it gives shorter computation time by occupying reasonable area. For example, for $GF(2^{163})$, the TIT algorithm using a hybrid-double multiplier with $d = 21$ computes an inversion in 96.2 ns and occupies 131,176 μm^2 silicon area. The inverter using the IT algorithm and a single multiplier with $d = 55$ occupies almost the same area and computes an inversion in 132.6 ns. Therefore, the proposed scheme is about 28 percent faster while using the same silicon area.

7 Conclusions

We presented the TIT algorithm for inversions over binary fields. It is a new variation of the IT algorithm that can efficiently use double multiplications. We also designed an inverter architecture using a hybrid-double multiplier that exploits this algorithm and showed that it computes inversions over the NIST fields (excluding $GF(2^{233})$) faster than architectures using the IT algorithm with single or hybrid-double multipliers.

Because double multipliers compute two multiplications simultaneously, our work can be seen as an effort to parallelize inversions. The previous work [14] did not reduce the number of multiplications on the critical path. Instead, they exploited square roots in $GF(2^m)$ with polynomial basis and effectively run two IT algorithms (based on squarings and square roots) in parallel [14]. Consequently, their approach requires fast multipliers (bit-parallel multipliers were used in [14]) because only then squarings and square roots play a significant role in latency. Our work is more general because it focused on reducing the latency of multiplications which typically dominate inversions.

TABLE 3
ASIC Synthesis Results of the Inverters for $GF(2^{163})$, $GF(2^{283})$, $GF(2^{409})$, and $GF(2^{571})$ Using 65-nm CMOS

d	q	$q+1$	IT (single DL-PIPO Multiplier)				IT (Hybrid-double Multiplier)				TIT (Hybrid-double Multiplier)			
			Latency [cycles]	CPD [ns]	Time [ns]	Area [μm^2]	Latency [cycles]	CPD [ns]	Time [ns]	Area [μm^2]	Latency [cycles]	CPD [ns]	Time [ns]	Area [μm^2]
$m = 163$, $T = 4$														
11	15	16	146	0.94	137.2	32,542	155	1.32	204.6	72,923	87	1.32	114.8	73,171
21	8	9	83	1.61	133.6	56,538	92	1.85	170.2	130,928	52	1.85	96.2	131,176
33	5	6	56	2.28	127.7	84,441	65	2.25	146.2	199,045	37	2.25	83.2	199,293
41	4	5	47	2.67	125.5	103,421	56	2.87	160.7	245,971	32	2.87	91.8	246,219
55	3	4	38	3.49	132.6	136,100	47	3.61	169.7	325,567	27	3.61	97.4	325,815
$m = 283$, $T = 6$														
15	19	20	222	1.33	295.3	87,455	233	1.65	384.5	227,201	170	1.65	280.5	272,736
29	10	11	123	2.32	285.4	161,584	134	2.55	341.7	423,072	98	2.55	249.9	468,607
36	8	9	101	2.77	279.8	197,657	112	3.12	347.2	521,131	82	3.12	254.2	566,667
$m = 409$, $T = 4$														
10	29	30	332	0.96	318.7	71,115	312	1.38	430.6	165,346	219	1.38	302.2	165,668
18	16	17	189	1.42	268.4	121,151	182	1.68	305.8	280,993	128	1.68	215.1	281,315
23	13	14	156	1.65	257.5	152,820	152	1.91	290.1	354,699	107	1.91	204.4	355,021
$m = 571$, $T = 10$														
13	22	23	301	1.32	397.3	212,366	314	1.95	612.3	608,136	194	1.95	378.3	608,603
22	13	14	184	2.12	390.1	344,012	197	2.44	480.7	1,004,246	122	2.44	297.7	1,004,713
26	11	12	158	2.33	368.2	407,086	171	2.63	449.8	1,181,258	106	2.63	278.8	1,181,725

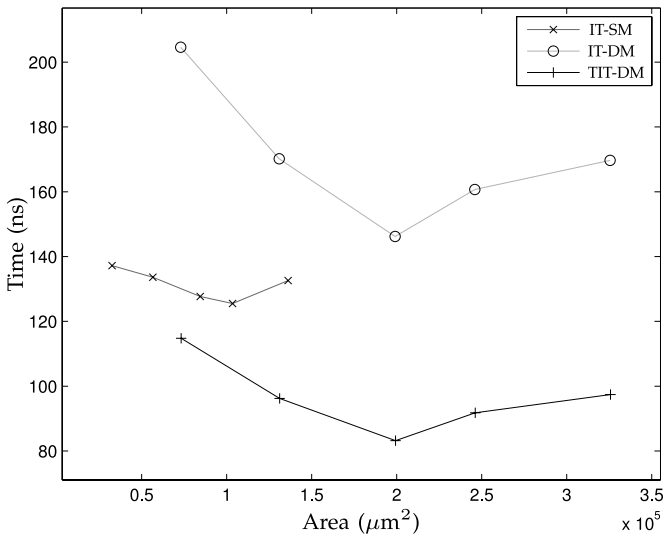

Fig. 5. An area-time plot of the ASIC results for $GF(2^{163})$ given in Table 3. IT-SM (single multiplication) is the architecture from Section 5.1, IT-DM (double multiplication) is from Section 5.3, and TIT-DM is from Section 5.2.

Although we considered only normal basis in this paper, the TIT algorithm is generic in the sense that it could be used also for $GF(2^m)$ with polynomial basis. We are not aware of double multipliers existing for polynomial basis and, hence, we encourage efforts to search efficient architectures for them. If efficient double multipliers were available, then the TIT algorithm could be used as such and the architecture of Fig. 4 could be straightforwardly adapted for polynomial basis.

We showed that a hybrid-double multiplier can reduce the latency of systems requiring inversions in binary fields, such as point multiplications on binary elliptic curves. The hybrid-double multipliers can be employed to reduce the latency of point addition and point doubling for different forms of elliptic curves such as binary Edwards, Koblitz, and generalized Hessian curves where high level of parallelization is not possible due to data dependencies [19]. Therefore, the proposed architecture for inversion can be efficiently employed in the ECC crypto-processors based on these curves.

In addition to inversions, systems using finite fields also need other operations, typically, additions, squarings, and multiplications. The architecture of Fig. 4 could be generalized with minor modifications into a field arithmetic unit supporting also these operations. Such a field arithmetic unit could be used in high-performance processors for elliptic curve cryptography.

Acknowledgments

The work of Kimmo Järvinen was funded by the Academy of Finland (the postdoctoral researcher's project #138358). Parts of the work by R. Azarderakhsh was done when he was working as a postdoctoral research fellow at Center for Applied Cryptographic Research, University of Waterloo, Canada.

References

[1] C.C. Wang, T.K. Truong, H.M. Shao, L.J. Deutsch, J.K. Omura, and I.S. Reed, "VLSI Architectures for Computing Multiplications and Inverses in $GF(2^m)$," *IEEE Trans. Computers*, vol. C-34, no. 8, pp. 709-717, Aug. 1985.

[2] T. Itoh and S. Tsujii, "A Fast Algorithm for Computing Multiplicative Inverses in $GF(2^m)$ Using Normal Bases," *Information and Computation*, vol. 78, no. 3, pp. 171-177, Sept. 1988.

[3] G.B. Agnew, R.C. Mullin, I.M. Onyszchuk, and S.A. Vanstone, "An Implementation for a Fast Public-Key Cryptosystem," *J. Cryptology*, vol. 3, no. 2, pp. 63-79, Jan. 1991.

[4] B. Sunar and Ç.K. Koç, "An Efficient Optimal Normal Basis Type II Multiplier," *IEEE Trans. Computers*, vol. 50, no. 1, pp. 83-87, Jan. 2001.

[5] S. Kwon, K. Gaj, C.H. Kim, and C.P. Hong, "Efficient Linear Array for Multiplication in $GF(2^m)$ Using a Normal Basis for Elliptic Curve Cryptography," *Proc. Sixth Int'l Workshop Cryptographic Hardware and Embedded Systems*, pp. 76-91, 2004.

[6] R. Azarderakhsh and A. Reyhani-Masoleh, "Low Complexity Multiplier Architectures for Single and Hybrid-Double Multiplications in Gaussian Normal Bases," *IEEE Trans. Computers*, vol. 62, no. 4, pp. 744-757, Apr. 2013.

[7] R. Lidl and H. Niederreiter, *Finite Fields*, second ed. Cambridge Univ. Press, 1997.

[8] D.W. Ash, I.F. Blake, and S.A. Vanstone, "Low Complexity Normal Bases," *Discrete Appl. Math.*, vol. 25, no. 3, pp. 191-210, 1989.

[9] *IEEE Standard Specifications for Public-Key Cryptography*, Std 1363-2000, Jan. 2000.

[10] National Institute of Standards and Technology (NIST), *Digital Signature Standard (DSS), Federal Information Processing Standard*, FIPS PUB 186-3, Jun. 2009.

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

[12] A. Reyhani-Masoleh, "Efficient Algorithms and Architectures for Field Multiplication Using Gaussian Normal Bases," *IEEE Trans. Computers*, vol. 55, no. 1, pp. 34-47, Jan. 2006.

[13] N. Takagi, J. Yoshiki, and K. Takagi, "A Fast Algorithm for Multiplicative Inversion in $GF(2^m)$ Using Normal Basis," *IEEE Trans. Computers*, vol. 50, no. 5, pp. 394-398, May 2001.

[14] F. Rodríguez-Henríquez, G. Morales-Luna, N.A. Saqib, and N. Cruz-Cortés, "Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials," *Designs Codes and Cryptography*, vol. 45, no. 1, pp. 19-37, Oct. 2007.

[15] G.-L. Feng, "A VLSI Architecture for Fast Inversion in $GF(2^m)$," *IEEE Trans. Computers*, vol. 38, no. 10, pp. 1383-1386, Oct. 1989.

[16] L. Gao and G.E. Sobelman, "Improved VLSI Designs for Multiplication and Inversion in $GF(2^M)$ over Normal Bases," *Proc. 13th Ann. IEEE Int'l ASIC/SOC Conf.*, pp. 97-101, Sept. 2000.

[17] K. Järvinen and J. Skyttä, "Fast Point Multiplication on Koblitz Curves: Parallelization Method and Implementations," *Microprocessors and Microsystems*, vol. 33, no. 2, pp. 106-116, Mar. 2009.

[18] Synopsys, Inc., www.synopsys.com, 2014.

[19] R. Azarderakhsh and A. Reyhani-Masoleh, "Efficient FPGA Implementations of Point Multiplication on Binary Edwards and Generalized Hessian Curves Using Gaussian Normal Basis," *IEEE Trans. VLSI Syst.*, vol. 20, no. 8, pp. 1453-1466, Aug. 2012.