Parallel Itoh-Tsujii Multiplicative Inversion Algorithm for a Special Class of Trinomials

Francisco Rodríguez-Henríquez[1], Guillermo Morales-Luna[1], Nazar A. Saqib[2] and Nareli Cruz-Cortés[1]
(1) Computer Science Section, Electrical Engineering Department
Centro de Investigación y de Estudios Avanzados del IPN
Av. Instituto Politécnico Nacional No. 2508, México D.F.
{francisco, gmorales, nareli}@cs.cinvestav.mx
(2) Centre for Cyber Technology and Spectrum Management
(CCT & SM), NUST, Islamabad-Pakistan.
nabbas-cctsm@nust.edu.pk

Abstract

In this contribution, we derive a novel parallel formulation of the standard Itoh-Tsujii algorithm for multiplicative inverse computation over $GF(2^m)$. The main building blocks used by our algorithm are: field multiplication, field squaring and field square root operators. It achieves its best performance when using a special class of irreducible trinomials, namely, $P(X) = X^m + X^k + 1$, with m and k odd numbers and when implemented in hardware platforms. Under these conditions, our experimental results show that our parallel version of the Itoh-Tsujii algorithm yields a speedup of about 30% when compared with the standard version of it. Implemented in a Virtex 3200E FPGA device, our design is able to compute multiplicative inversion over $GF(2^{193})$ after 20 clock cycles in about 0.94μS.

1 Introduction

Binary extension finite fields $GF(2^m)$ are extensively used in many modern applications, such as public and secret key cryptosystems, error-correcting codes, etc. Perhaps one of the most important applications for this class of finite fields is elliptic Curve Cryptography (ECC), since high performance ECC implementations directly rely on the efficiency in the computation of the underlying finite field arithmetic operations.

Among customary finite field arithmetic operations, namely, addition, subtraction, multiplication and inversion of nonzero elements, the computation of the later is the most time-consuming one. Multiplicative inversion computation of a nonzero element $a \in GF(2^m)$ is defined as the process of finding the unique element $a^{-1} \in GF(2^m)$ such that $a \cdot a^{-1} = 1$.

Several algorithms for computing the multiplicative inverse in $GF(2^m)$ have been proposed in literature [1–8]. In [4], multiplicative inverse is computed using an improved modification of the extended Euclidean algorithm called *almost inverse algorithm*. That iterative algorithm can compute the multiplicative inverse in approximately $2m$ clock cycles [4]. In [6] an architecture able to compute the Montgomery multiplicative inverse for both, $GF(p)$, for a prime p, and $GF(2^m)$ on a unified-field hardware platform was proposed.

Based on Fermat's Little Theorem (FLT) and using an ingenious re-arrangement of the required field operations, the *Itoh-Tsujii Multiplicative Inverse Algorithm* (ITMIA) was presented in [1]. Originally, ITMIA was proposed to be applied over binary extension fields with *normal basis* field element representation. Since its publication however, several improvements and variations of it have been reported [2, 3, 5, 7, 8], showing that it can be used with other field element representations too.

Unfortunately enough, cryptographic designers have historically shown some resistance to use FLT-related techniques for computing multiplicative inverses when using polynomial basis representation. This phenomenon is probably due to three frequent misconceptions:

1. Computing multiplicative inverses by using FLT-related techniques is inefficient as those methods require many field multiplication and squaring operations;

2. ITMIA is a competitive design option only when using normal basis representation and;

3. The recursive nature of the ITMIA algorithm makes the parallelization of that algorithm rather difficult if not impossible, forcing the implementation of the ITMIA procedure in a sequential manner.

In this contribution we hope to refute (at least partially) above opinions. First, we discuss how to combine the ITMIA standard procedure with the notion of *addition chains* [9] (it is noticed that this discussion closely follows the approach presented in [7]). We show that this technique constitutes a competitive option for computing multiplicative inverses in $GF(2^m)$ not only with normal basis but also with *polynomial (canonical)* field element representation.

Then, considering a special class of irreducible trinomials, namely, $P(X) = X^m + X^k + 1$, with m and k odd numbers, we derive a novel version of the Itoh-Tsujii algorithm which uses field multiplication, field squaring and field square root operators as main building blocks. We also show how this version of the algorithm can be parallelized when implemented in hardware platforms. Our experimental results show that the parallel version of the Itoh-Tsujii algorithm implementation yields a speedup of about 30% when compared with the standard version of it.

The rest of this paper is organized as follows. In Section 2 some important mathematical concepts are reviewed. In particular, we discuss how to calculate field square and field square root operations efficiently when working in a binary extension finite field generated by an irreducible trinomial of the form, $P(X) = X^m + X^k + 1$, with m and k odd numbers. In Section 3, the standard version of the Itoh-Tsujii algorithm combined with the concept of addition chains is presented. We give a rigorous complexity analysis of that algorithm. Then, in Section 4, a novel parallel formulation of the ITMIA procedure is presented, including a rigorous complexity analysis of this procedure. In Section 5, design details of our algorithm hardware implementation are discussed in detail. In Section 6 we summarize the main performance figures of the block designs and we also provide a comparison with other reported designs found in the open literature. Finally, in Section 7 some conclusions remarks are drawn.

2 Mathematical Preliminaries

2.1 Binary Field Arithmetic

Let $\mathbb{K}$ be a finite field and let $P(X) \in \mathbb{K}[X]$ be a polynomial with coefficients in $\mathbb{K}$. $P(X)$ is *irreducible* over $\mathbb{K}$ if whenever $P(X) = Q(X)R(X)$, either $Q(X)$ or $R(X)$ is an unit in $\mathbb{K}[X]$, i.e. it is a constant polynomial.

Let $GF(2)$ be the prime field of characteristic 2: $GF(2) = \{0, 1\}$, addition is XOR and multiplication is the logical AND. Let $P(X) \in GF(2)[X]$ be an irreducible polynomial of degree $m > 1$, $m \in \mathbb{N}$, and let α be a root of $P(X)$ in a finite extension of $GF(2)$. Then the Galois field $GF(2^m)$ is isomorphic to the finite extension $GF(2)[\alpha]$, it has 2^m elements and the powers of α form a basis of $GF(2^m)$ over $GF(2)$. The set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ is called the *polynomial basis* of $GF(2^m)$ corresponding to α. Thus, for each $a \in GF(2^m)$ there exists a polynomial $A(X) \in GF(2)[X]$ of degree at most $m-1$ such that $a = A(\alpha)$ and this is the *polynomial representation* of a. Addition in $GF(2^m)$ is performed by adding corresponding coefficients using polynomial representations. Multiplication corresponds to polynomial multiplication reduced modulus the irreducible polynomial $P(X)$: If $a \in GF(2^m)$ is represented by the polynomial $A(X)$ and $b \in GF(2^m)$ is represented by the polynomial $B(X)$ then $c = ab$ is represented by the polynomial $C(X) = A(X)B(X) \bmod P(X)$. The multiplicative group of $GF(2^m)$ is cyclic. If the irreducible polynomial $P(X)$ is also primitive, then any of its roots α is a generator of the multiplicative group of $GF(2^m)$ and has order $2^m - 1$. Thus for any nonzero $a \in GF(2^m)$, if $i \equiv j \bmod (2^m - 1)$ then $a^i = a^j$ in $GF(2^m)$. Since the field characteristic is 2, the squaring operator is linear:

$$a = A(\alpha) = \sum_{j=0}^{m-1} a_j \alpha^j \ \Rightarrow \ a^2 = \sum_{j=0}^{m-1} a_j \alpha^{2j} = A(\alpha^2). \tag{1}$$

Let us assume that the irreducible polynomial is a trinomial: $P(X) = X^m + X^n + 1$, with $n \leq m/2$. Then, for all $i \geq 0$, $X^{m+i} = (X^n + 1)X^i \bmod P(X)$. Thus whenever $j \geq m/2$, there exists $i \geq 0$ such that $2j = m + i$ and $\alpha^{2j} = (\alpha^n + 1)\alpha^i$ in the field $GF(2^m)$, consequently the upper half of terms in a^2, according to eq. (1), can be calculated by additions and shift operations only. Namely,

$$m \equiv 0 \bmod 2 \ \Rightarrow \ a^2 = \sum_{j=0}^{\frac{m}{2}-1} a_{2j}\alpha^j + (\alpha^n + 1) \sum_{j=0}^{\frac{m}{2}-1} a_{\frac{m}{2}+j}\alpha^{2j} \tag{2}$$

$$m \equiv 1 \bmod 2 \ \Rightarrow \ a^2 = \sum_{j=0}^{\frac{m-1}{2}} a_{2j}\alpha^j + (\alpha^n + 1) \sum_{j=0}^{\frac{m-3}{2}} a_{\frac{m+1}{2}+j}\alpha^{2j+1} \tag{3}$$

Moreover, using just the representing polynomial, we can compute the reduction step $C(X) = A(X)^2 \bmod P(X)$ by adding four terms, B_1, B_2, B_3 and B_4 as,

$$\begin{aligned} C &= A'_{[0,m-1]} + A'_{[m,2m-1]} + A'_{[m,2m-1-n]}X^n \\ &\quad + \left(A'_{[2m-n,2m-1]} + A'_{[2m-n,2m-1]}X^n\right) \\ &= B_1 + B_2 + B_3 + B_4 \end{aligned} \tag{4}$$

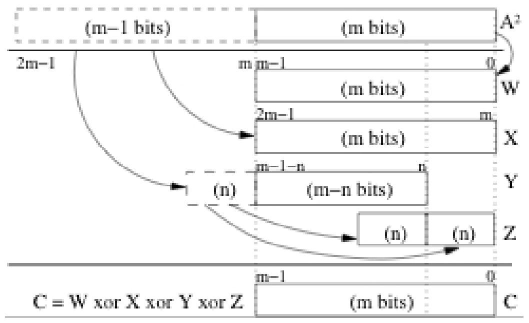

Figure 1: Reduction scheme.

where $A'_{[i,j]}$ denotes the list of coefficients in $A(X)$ with indexes ranging from i to j. The reduction procedure described in eq. (4) is shown schematically in Fig. 2.1. Notice that for those designs implemented in hardware platforms, the field squaring computation procedure just outlined can be instrumented by using XOR logic gates only. Nevertheless, the exact computational complexity of this arithmetic operation depends on the explicit form of m and the middle coefficient n. For instance, in the case of the irreducible trinomial $P(X) = X^{193} + X^{15} + 1$, field squaring operation can be computed as,

$$c_i = \begin{cases} a_{i/2} & i \text{ even, } i < 15, \\ a_{i/2} + a_{i/2+89} + a_{178+i/2} & i \text{ even, } 15 < i < 30, \\ a_{i/2} + a_{i/2+89} & i \text{ even, } i \geq 30, \\ a_{(m+i)/2} + a_{193-(15-i)/2} & i \text{ odd, } i < 15, \\ a_{(m+i)/2} & i \text{ odd, } i \geq 15. \end{cases} \tag{5}$$

(5) can be implemented at a cost of 96 two-input XOR gates and two XOR gate delays, $2T_x$.

A straightforward but rather expensive approach for computing the square root operator is based on Fermat's little theorem which establishes that for any nonzero element $a \in GF(2^m)$, the identity $a^{2^m} = a$ holds in the field $GF(2^m)$. Therefore, $\sqrt{a}$ may be computed as $\sqrt{a} = d = a^{2^{m-1}}$, with a computational cost of $m - 1$ field squarings [10]. A more efficient algorithm based on a refining of the Fermat's little theorem method just outlined was proposed in [11]. That method is based on the observation that $\sqrt{a}$ can be expressed in terms of the square root of the root element α. In the case that the irreducible polynomial is a trinomial, authors in [11] showed that the square root operator can be computed with some shift-left operations and one modular reduction.

In this contribution we outline an alternative method for computing $\sqrt{a}$, which is also based on the linearity property exhibited by the field squaring operation defined over binary extension fields. Considering that the squaring map $a \mapsto a^2$ is linear, if we denote by A_1 and A_2 the polynomial representation of a and a^2 respectively, there exists a square matrix M_2 of order $m \times m$ with coefficients in $GF(2)$ such that $A_2 = M_2 A_1$. The matrix M_2 can be calculated from eq's (2) and (3). If we are interested in calculating the square root $d = \sqrt{a} \in GF(2^m)$, with polynomial representation $D(\alpha)$, then we must have $M_2 D = A_1$, or $D = M_2^{-1} A_1$ which gives the alternative procedure for square root calculation.

For instance, for $P(X) = X^{193} + X^{15} + 1$, one can solve last matrix equation based on (5). The following set of equations are then obtained,

$$d_i = \begin{cases} a_{2i} & i < 8, \\ a_{2i} + a_{2i-15} & 8 \leq i < 97 \\ a_{2i-15} + a_{2i-193} & 97 \leq i < 104, \\ a_{2i-193} & 104 \leq i \end{cases} \tag{6}$$

Eq. (6) can be implemented at a cost of 96 two-input XOR gates and one T_x gate delay.

Let us now briefly examine the problem of field exponentiation computation. Let $a = A(\alpha) \in GF(2^m)$ be a nonzero element in the field and let e be an m-bit positive integer, i.e., $m = \lceil \log_2(e) \rceil$. Then the field exponentiation of the element a raised to the power e is represented by the polynomial

$$C(X) = A(X)^e \bmod P(X). \tag{7}$$

There exist many reported algorithms [9, 12] for the efficient computation of (7), being the *binary exponentiation algorithm* the most popular for hardware implementations. This is probably because of its simplicity and relatively good performance/cost benefit. The binary exponentiation algorithm (as in fact most exponentiation algorithms) utilizes modular multiplication and squaring as the most prominent building blocks needed to obtain the desired computation.

Next Section discusses how field exponentiation can be used for computing field multiplicative inverses.

3 Itoh-Tsujii Multiplicative Inversion Algorithm

In this Section we describe the Itoh-Tsujii Multiplicative Inversion Algorithm (IT-MIA). We start deriving a recursive sequence useful for finding multiplicative inverses. Then, we briefly discuss the concept of *addition chains*, which together with the aforementioned recursive sequence yield an efficient version of the original ITMIA procedure as it was presented in [1].

3.1 A Square-Then-Multiply Recursive Sequence of Field Elements

Since the multiplicative group of the Galois field $GF(2^m)$ is cyclic of order $2^m - 1$, for any nonzero element $a \in GF(2^m)$ we have $a^{-1} = a^{2^m-2}$. Clearly,

$$2^m - 2 = 2(2^{m-1} - 1) = 2\sum_{j=0}^{m-2} 2^j = \sum_{j=1}^{m-1} 2^j.$$

The right-most component of above equalities allow us to express the multiplicative inverse of a in two ways:

$$\left[a^{2^{m-1}-1}\right]^2 = a^{-1} = \prod_{j=1}^{m-1} a^{2^j} \tag{8}$$

Let us consider the sequence $\left(\beta_k(a) = a^{2^k-1}\right)_{k\in\mathbb{N}}$. Then, for instance,

$$\beta_0(a) = 1 \quad , \quad \beta_1(a) = a,$$

and from the first equality at (8), $[\beta_{m-1}(a)]^2 = a^{-1}$.

It is easy to see that for any two integers $k, j \geq 0$,

$$\beta_{k+j}(a) = \beta_k(a)^{2^j}\beta_j(a). \tag{9}$$

Namely

$$\begin{aligned}
\beta_{k+j}(a) &= a^{2^{k+j}-1} = \frac{a^{2^{k+j}}}{a} = \frac{\left(a^{2^k}\right)^{2^j}}{a} \\
&= \left(\frac{a^{2^k}}{a}\right)^{2^j} \frac{a^{2^j}}{a} = \left(a^{2^k-1}\right)^{2^j} a^{2^j-1} \\
&= \beta_k(a)^{2^j}\beta_j(a)
\end{aligned}$$

In particular, for $j = k$,

$$\beta_{2k}(a) = \beta_k(a)^{2^k}\beta_k(a) = \beta_k(a)^{2^k+1}. \tag{10}$$

Furthermore, we observe that this sequence is periodic of period m:

$$k_2 \equiv k_1 \bmod m \;\Rightarrow\; \beta_{k_2}(a) = \beta_{k_1}(a).$$

To see this, consider $k_2 = k_1 + nm$. Then, by eq. (9) and FLT,

$$\beta_{k_2}(a) = \beta_{k_2}(a)^{2^{nm}}\beta_{nm}(a) = \beta_{k_2}(a)^{2^{nm}} \cdot 1 = \beta_{k_2}(a).$$

Therefore, the sequence $\left(\beta_k(a)\right)_k$ is completely determined by its values corresponding to the indexes $k = 0, \ldots, m-1$.

As a final remark, notice that for any two integers k, j, by eq. (9):

$$\beta_k(a) = \beta_{(k-(m-j))+(m-j)}(a) = \beta_{k+j-m}(a)^{2^{m-j}}\beta_{m-j}(a).$$

Since the sequence of β's is periodic, and the rising to the power 2^m coincides with the identity in *GF*(2^m), we have

$$\beta_k(a) = \beta_{k+j}(a)^{2^{-j}}\beta_{m-j}(a). \tag{11}$$

Eq. (9) allows the calculation of a "current" $i(= k + j)$-th term as a recursive function of two previous terms, the k-th and the j-th in the sequence.

3.2 Addition Chains

Let us say that an *addition chain* for an integer $m - 1$ consists of a finite sequence of integers $U = (u_0, u_1, \ldots, u_t)$, and a sequence of integer pairs $V = ((k_1, j_1), \ldots, (k_t, j_t))$ such that $u_0 = 1$, $u_t = m - 1$, and whenever $1 \leq i \leq t$, $u_i = u_{k_i} + u_{j_i}$.

Example 3.1. *Consider the case* $e = m-1 = 193-1 = 192 = (11000000)_2$. *Then, a binary addition chain with length* $t = 8$ *for that* e *is,*

$$\begin{array}{lllllllllll} U & = & (& 1, & 2, & 4, & 8, & 16, & 32, & 64, & 128, & 192) \\ V & = & (& & (0,0), & (1,1), & (2,2), & (3,3), & (4,4), & (5,5), & (6,6), & (6,7)) \end{array}$$

i.e. the associated sequence is governed by the rule, $u_i = u_{i-1} + u_{i-1} = 2u_{i-1}$ *for all but the final value which is obtained using* $u_t = u_{t-1} + u_{t-2}$.

Another addition chain, also with length $t = 8$, is

$$\begin{array}{rcllllllllll} U &=& (& 1, & 2, & 3, & 6, & 12, & 24, & 48, & 96, & 192) \\ V &=& (& & (0,0), & (0,1), & (2,2), & (3,3), & (4,4), & (5,5), & (6,6), & (7,7)) \end{array}$$

i.e. for all $i \neq 2$ the combinatorial rule is $u_i = u_{i-1} + u_{i-1} = 2u_{i-1}$, while $u_2 = u_0 + u_1$. □

The concept of addition chains leads us to a natural way to generalize the Itoh-Tsujii Algorithm, by using an addition chain for $m - 1$ and relations (8) and (9) to compute $a^{-1} = [\beta_{m-1}(a)]^2$.

3.3 ITMIA Algorithm

Let a be any arbitrary nonzero element in the field $GF(2^m)$. Let us consider an addition chain U of length t for $m - 1$ and its associated sequence V. Then the multiplicative inverse $a^{-1} \in GF(2^m)$ of a can be found by repeatedly applying eq's. (9) and/or (10). Hence, given $\beta_{u_0}(a) = a^{2^1-1} = a$, for each $u_i, 1 \leq i \leq t$, compute

$$\left[\beta_{u_{i_1}}(a)\right]^{2^{u_{i_2}}} \beta_{u_{i_2}}(a) = \beta_{u_{i_2}+u_{i_1}}(a) = \beta_{u_i}(a) = a^{2^{u_i}-1}$$

A final squaring step yields the required result since,

$$[\beta_{u_t}(a)]^2 = \left(a^{2^{m-1}-1}\right)^2 = (a^{2^m-2}) = a^{-1}.$$

Fig. 2 shows an algorithm that iteratively computes all the $\beta_{u_i}(a)$ coefficients in the exact order stipulated by the addition chain U as discussed above.

We assess the computational complexity of the algorithm shown in Fig. 2 as follows. The algorithm performs t iterations (where t is the length of the addition chain U) and one field multiplication per iteration. Thus, we conclude that a total of t field multiplication computations are required. On the other hand, notice that at each iteration i, a total of $2^{u_{i_2}}$ field squarings are performed. Notice also that by definition, the addition chain guarantees that for each $u_i, 1 \leq i \leq t$, the relation $u_{i_2} = u_i - u_{i_1}$ holds. Hence, one can show by induction that the total number of field squaring operations performed right after the execution of the i-th iteration is $u_i - 1$. Therefore, at the end of the final iteration t, a total of $u_t - 1 = m - 2$ squaring operations have been performed. This, together with the final squaring operation, yield a total of $m - 1$ field squaring computations.

```
Input:  An irreducible polynomial $P(X)$ of degree $m$, An
        element $a \in GF(2^m)$, an addition chain $U$ of
        length $t$ for $m-1$ and its associated sequence $V$
Output: $a^{-1} \in GF(2^m)$

Procedure ITMIA($P(X), a, \{U, V\}$) {
1. $\beta_{u_0}(a) = a$;
2. for $i$ from 1 to $t$ do
3.     $\beta_{u_i}(a) = \left[\beta_{u_{i_1}}(a)\right]^{2^{u_{i_2}}} \cdot \beta_{u_{i_2}}(a) \bmod P(X)$;
4. return ($\beta^2_{u_t}(a) \bmod P(X)$);
}
```

Figure 2: Multiplicative Inversion Addition-Chain Itoh-Tsujii Algorithm

Summarizing, the algorithm of Fig. 2 can find the multiplicative inverse of any nonzero element of the field using exactly,

$$\begin{aligned} \#\textit{Multiplications} &= t; \\ \#\textit{Squarings} &= m-1. \end{aligned} \tag{12}$$

Example 3.2. *Let us consider the binary field $GF(2^{193})$ using the irreducible trinomial $P(X) = X^{193} + X^{15} + 1$. Let $a \in GF(2^{193})$ be an arbitrary nonzero field element. Then, using the addition chain of Example 3.1, the algorithm of Fig. 2 would compute the sequence of $\beta_{u_i}(a)$ coefficients as shown in Table 3.3. Once again, notice that after having computed the coefficient $\beta_{u_8}(a)$, the only remaining step is to obtain a^{-1} which can be achieved as $a^{-1} = \beta^2_{u_8}(a)$.* □

4 A Parallel Version of the Itoh-Tsujii Algorithm

In this Section we reformulate the ITMIA algorithm in terms of square root operations rather than field squarings. Then, we show how to combine both versions of the Itoh-Tsujii algorithm in order to obtain a parallel version of it.

Table 1: $\beta_i(a)$ Coefficient Generation for m-1=192

i	u_i	rule	$\left[\beta_{u_{i_1}}(a)\right]^{2^{u_{i_2}}}(a)\cdot\beta_{u_{i_2}}(a)$	$\beta_{u_i}(a) = a^{2^{u}_{i}-1}$
0	1	–	–	$\beta_{u_0}(a) = a^{2^1-1}$
1	2	$2u_{i-1}$	$[\beta_{u_0}(a)]^{2^{u_0}}(a)\cdot\beta_{u_0}(a)$	$\beta_{u_1}(a) = a^{2^2-1}$
2	3	$u_{i-1}+u_{i-2}$	$[\beta_{u_1}(a)]^{2^{u_0}}(a)\cdot\beta_{u_0}(a)$	$\beta_{u_2}(a) = a^{2^3-1}$
3	6	$2u_{i-1}$	$[\beta_{u_2}(a)]^{2^{u_2}}(a)\cdot\beta_{u_2}(a)$	$\beta_{u_3}(a) = a^{2^6-1}$
4	12	$2u_{i-1}$	$[\beta_{u_3}(a)]^{2^{u_3}}(a)\cdot\beta_{u_3}(a)$	$\beta_{u_4}(a) = a^{2^{12}-1}$
5	24	$2u_{i-1}$	$[\beta_{u_4}(a)]^{2^{u_4}}(a)\cdot\beta_{u_4}(a)$	$\beta_{u_5}(a) = a^{2^{24}-1}$
6	48	$2u_{i-1}$	$[\beta_{u_5}(a)]^{2^{u_5}}(a)\cdot\beta_{u_5}(a)$	$\beta_{u_6}(a) = a^{2^{48}-1}$
7	96	$2u_{i-1}$	$[\beta_{u_6}(a)]^{2^{u_6}}(a)\cdot\beta_{u_6}(a)$	$\beta_{u_7}(a) = a^{2^{96}-1}$
8	192	$2u_{i-1}$	$[\beta_{u_7}(a)]^{2^{u_7}}(a)\cdot\beta_{u_7}(a)$	$\beta_{u_8}(a) = a^{2^{192}-1}$

4.1 A Square Root-Then-Multiply Recursive Sequence of Field Elements

For any nonzero $a \in GF(2^m)$, we have $a^{\frac{1}{2}} = a^{2^{m-1}}$, and

$$\forall j \leq m-1 : \quad a^{2^{-j}} = a^{2^{m-j}} \tag{13}$$

just because $2^{j(m-1)} \equiv 2^{m-j} \bmod (2^m-1)$. Using (8) and (13), we obtain

$$a^{-1} = \prod_{j=m-1}^{1} a^{2^{m-j}} = \prod_{j=m-1}^{1} a^{2^{-j}} = a^{\sum_{j=m-1}^{1} 2^{-j}}$$

(observe that all indexes are varying in descending order). Noticing now that,

$$\sum_{j=m-1}^{1} 2^{-j} = \frac{1-\left(\frac{1}{2}\right)^m}{1-\left(\frac{1}{2}\right)} - 1 = \frac{2^m-1}{2^{m-1}} - 1 = \frac{2^{m-1}-1}{2^{m-1}},$$

we get,

$$a^{-1} = a^{\frac{2^{m-1}-1}{2^{m-1}}}$$

In an analogous way as we did in Subection 3.1 for the sequence $(\beta_k(a))_{k\in\mathbb{N}}$, let us define now the sequence $\left(\gamma_k(a) = a^{1-2^{-k}}\right)_{k\in\mathbb{N}}$. First of all, let us remark that, from (13), $\forall k \in \mathbb{N}$:

$$\gamma_k(a) = a^{1-2^{-k}} = a \cdot \left[a^{2^{-k}}\right]^{-1} = a \cdot \left[a^{2^{m-k}}\right]^{-1} = [\beta_{m-k}(a)]^{-1}. \tag{14}$$

In particular, for $k = m-1$,

$$a^{-1} = \gamma_{m-1}(a). \tag{15}$$

As in (9), for any two integers $k, j \geq 0$,

$$\gamma_{k+j}(a) = \gamma_k(a)^{2^{-j}} \gamma_j(a). \tag{16}$$

and, as in (10), for $j = k$,

$$\gamma_{2k}(a) = \gamma_k(a)^{2^{-k}} \gamma_k(a) = \gamma_k(a)^{2^{-k}+1} \tag{17}$$

The sequence $(\gamma_k(a))_{k\in\mathbb{N}}$ is also periodic with period m. Moreover, for any two integers k, j, the analogous of (11) would be,

$$\gamma_k(a) = \gamma_{k+j}(a)^{2^j} \gamma_{m-j}(a).$$

Furthermore, using eq's. (15), (16) and (13), $\forall k \leq m-2$:

$$a^{-1} = \gamma_{m-1}(a) = \gamma_{m-1-k}(a)^{2^{-k}} \gamma_k(a) = \left[\beta_{k+1}(a)^{2^{-k}} \beta_{m-k}(a)\right]^{-1}.$$

Let us remark also that if k, j are such that $k + j = m - j$, then, according with (9) and (14),

$$\begin{aligned}
\beta_k(a)\left(\beta_j(a)^{2^k}\gamma_j(a)\right) &= \left(\beta_j(a)^{2^k}\beta_k(a)\right)\gamma_j(a) \\
&= \beta_{k+j}(a)\,\gamma_j(a) \\
&= \beta_{m-j}(a)\,\gamma_j(a) \\
&= 1
\end{aligned}$$

and consequently $\beta_k(a)^{-1} = \beta_j(a)^{2^k}\gamma_j(a)$. In summary

$$m \equiv k \bmod 2 \implies \beta_k(a)^{-1} = \beta_{\frac{m-k}{2}}(a)^{2^k}\gamma_{\frac{m-k}{2}}(a), \tag{18}$$

which in turn gives

$$k \equiv 0 \bmod 2 \implies \gamma_k(a) = \beta_{\frac{k}{2}}(a)^{2^{m-k}}\gamma_{\frac{k}{2}}(a).$$

Thus, in particular by letting $k = 1$ in eq. (18) we get,

$$m \equiv 1 \bmod 2 \implies a^{-1} = \beta_{\frac{m-1}{2}}(a)^2 \gamma_{\frac{m-1}{2}}(a) = \left[a^{2^{\frac{m-1}{2}}-1}\right]^2 a^{1-2^{-\frac{m-1}{2}}}.$$

Our modification of Itoh-Tsujii algorithm uses now an addition chain for $\frac{m-1}{2}$ and relations (16) and (17) to compute $a^{-1} = \gamma_{m-1}(a)$. We summarize below what we consider the main result of this paper,

Theorem 4.1. *Let $a \in GF(2^m)$ be an arbitrary nonzero field element, with m odd. Then, its multiplicative inverse, a^{-1}, can be found as,*

$$a^{-1} = \left[\beta_{\{\frac{m-1}{2}\}}(a)\right]^2 \gamma_{\{\frac{m-1}{2}\}}(a) = \left[a^{(2^{\frac{m-1}{2}}-1)}\right]^2 \cdot a^{1-2^{-\frac{(m-1)}{2}}}. \quad (19)$$

Example 4.2. *Consider the smallest binary finite field, namely, GF$(2^1) = \{0, 1\}$, then according to (19) the multiplicative inverse of the only nonzero field element would be,*

$$\begin{aligned} 1^{-1} &= \left[\beta_{\{\frac{1-1}{2}\}}(1)\right]^2 \gamma_{\{\frac{1-1}{2}\}}(1) = \left[1^{(2^{\frac{1-1}{2}}-1)}\right]^2 \cdot 1^{1-2^{-\frac{(1-1)}{2}}} \\ &= [1^0]^2 \cdot 1^0 = 1. \end{aligned}$$

□

Example 4.3. *Consider the binary finite field GF(2^3), then according to (19) the multiplicative inverse of an arbitrary nonzero field element would be,*

$$\begin{aligned} a^{-1} &= \left[\beta_{\{\frac{3-1}{2}\}}(a)\right]^2 \gamma_{\{\frac{3-1}{2}\}}(a) = \left[a^{(2^{\frac{3-1}{2}}-1)}\right]^2 \cdot a^{1-2^{-\frac{(3-1)}{2}}} \\ &= a^2 \cdot a^{1-2^{-1}} = a^2 \cdot a \cdot \left[a^{2^{-1}}\right]^{-1} = a^3 \cdot \left[a^{2^{3-1}}\right]^{-1} = a^3 \cdot \left[a^4\right]^{-1} = a^{-1}. \end{aligned}$$

Notice that in virtue of (13), the equality $a^{2^{-1}} = a^{2^{3-1}}$ used above holds. □

In the rest of this Section we describe how to use (19) in real applications, where binary extension fields GF(2^m) with m a large odd integer, are required.

4.2 Square Root ITMIA

Let a be any arbitrary nonzero element in the field $GF(2^m)$. Let us consider an addition chain U of length t for $m-1$ and its associated sequence V. Then the multiplicative inverse of a, $a^{-1} \in GF(2^m)$, can be found by repeatedly applying eq's. (16) and (17). Hence, given $\gamma_{u_0}(a) = a^{1-2^{-1}} = \sqrt{a}$, for each $u_i, 1 \le i \le t$, compute

$$\left[\gamma_{u_{i_1}(a)}\right]^{2^{-u_{i_2}}} \gamma_{u_{i_2}}(a) = \gamma_{u_{i_2}+u_{i_1}}(a) = \gamma_{u_i}(a) = a^{1-2^{-u_i}}$$

Where $\gamma_{\{u_t=m-1\}} = a^{1-2^{-(m-1)}} = a^{-1}$ gives the required result.

Fig. 3 shows an algorithm that iteratively computes all the $\gamma_{u_i}(a)$ coefficients in the exact order stipulated by the addition chain U as discussed above. We assess the computational complexity of the algorithm shown in Fig. 3 as follows. The algorithm performs one field multiplication in each of algorithm's t iterations, yielding a total of t field multiplication computations required. Furthermore, at each iteration i, a total of $2^{u_{i_2}}$ field square roots are performed. Since by definition, the addition chain guarantees that for each $u_i, 1 \le i \le t$, the relation $u_{i_2} = u_i - u_{i_1}$ holds, one can show that the total number of field square root operations performed right after the execution of the i-th iteration is $u_i - 1$. Therefore, a total of $u_t - 1 = m-2$ square root operations must be performed. This, together with the initial square root operation, yield a total of $m-1$ field square root computations.

Summarizing, the algorithm of Fig. 3 can find the inverse of any nonzero element of the field using exactly,

$$\begin{aligned} \#\textit{Multiplications} &= t; \\ \#\textit{Square root} &= m-1. \end{aligned} \tag{20}$$

Example 4.4. *Following with our running example, let us consider the binary field $GF(2^{193})$ generated using the irreducible trinomial $P(X) = X^{193} + X^{15} + 1$. Let $a \in GF(2^{193})$ be an arbitrary nonzero field element. Then, the algorithm of Fig. 3 would compute the sequence of $\gamma_{u_i}(a)$ coefficients as shown in Table 4.2. The multiplicative inverse is given as $\gamma_{u_8} = a^{-1}$.* □

4.3 Itoh-Tsujii Algorithm: Parallel Version

We can obtain a parallel version of the ITMIA algorithm, by executing in parallel algorithms of Figures 2 and 3, respectively. Notice that in virtue of (19), we just need to obtain the coefficients, $\beta_{u_{t-1}}$ and $\gamma_{u_{t-1}}$.

```
Input:   An irreducible polynomial $P(X)$ of degree $m$, An
         element $a \in GF(2^m)$, an addition chain $U$ of
         length $t$ for $m-1$ and its associated sequence $V$
Output: $a^{-1} \in GF(2^m)$

Procedure SquareRoot_ITMIA($P(X), a, \{U, V\}$) {
1. $\gamma_{u_0}(a) = a^{1-2^{-1}} = \sqrt{a}$;
2. for $i$ from 1 to $t$ do
3.     $\gamma_{u_i}(a) = \left[\gamma_{u_{i_1}}(a)\right]^{2^{-u_{i_2}}} \cdot \gamma_{u_{i_2}}(a) \bmod P(X)$;
4. return ($\gamma_{u_t}(a) \bmod P(X)$);
}
```

Figure 3: Multiplicative Inversion Addition-Chain Itoh-Tsujii Algorithm

Table 2: $\gamma_i(a)$ Coefficient Generation for m-1=192

i	u_i	rule	$\left[\gamma_{u_{i_1}}(a)\right]^{2^{-u_{i_2}}}(a) \cdot \gamma_{u_{i_2}}(a)$	$\gamma_{u_i}(a) = a^{1-2^{-u_i}}$
0	1	–	–	$\gamma_{u_0}(a) = a^{1-2^{-1}}$
1	2	$2u_{i-1}$	$[\gamma_{u_0}(a)]^{2^{-u_0}}(a) \cdot \gamma_{u_0}(a)$	$\gamma_{u_1}(a) = a^{1-2^{-2}}$
2	3	$u_{i-1} + u_{i-2}$	$[\gamma_{u_1}(a)]^{2^{-u_0}}(a) \cdot \gamma_{u_0}(a)$	$\gamma_{u_2}(a) = a^{1-2^{-3}}$
3	6	$2u_{i-1}$	$[\gamma_{u_2}(a)]^{2^{-u_2}}(a) \cdot \gamma_{u_2}(a)$	$\gamma_{u_3}(a) = a^{1-2^{-6}}$
4	12	$2u_{i-1}$	$[\gamma_{u_3}(a)]^{2^{-u_3}}(a) \cdot \gamma_{u_3}(a)$	$\gamma_{u_4}(a) = a^{1-2^{-12}}$
5	24	$2u_{i-1}$	$[\gamma_{u_4}(a)]^{2^{-u_4}}(a) \cdot \gamma_{u_4}(a)$	$\gamma_{u_5}(a) = a^{1-2^{-24}}$
6	48	$2u_{i-1}$	$[\gamma_{u_5}(a)]^{2^{-u_5}}(a) \cdot \gamma_{u_5}(a)$	$\gamma_{u_6}(a) = a^{1-2^{-48}}$
7	96	$2u_{i-1}$	$[\gamma_{u_6}(a)]^{2^{-u_6}}(a) \cdot \gamma_{u_6}(a)$	$\gamma_{u_7}(a) = a^{1-2^{-96}}$
8	192	$2u_{i-1}$	$[\gamma_{u_7}(a)]^{2^{-u_7}}(a) \cdot \gamma_{u_7}(a)$	$\gamma_{u_8}(a) = a^{1-2^{-192}}$

Example 4.5. *Let us consider once again the binary field GF(2^{193}) using the irreducible trinomial $P(X) = X^{193} + X^{15} + 1$. We can reuse coefficients β_7 and $\hat{\beta}_7$ defined in Tables 3.3 and 4.2, respectively. Then the multiplicative inverse of a can be found as, $a^{-1} = \beta^2{}_{u_7}\gamma_{u_7}$.* □

Since both coefficients above can be computed in parallel, this version of the Itoh-Tsujii algorithm will show a significant saving in time performance as it is discussed in the next Section.

5 Reconfigurable Hardware Architecture for Multiplicative Inversion in *GF*(2^{193})

In this Section, a description of our proposed architecture in reconfigurable hardware is presented. Field squarer, square root and Multiplier are the three most prominent building blocks for performing inversion in *GF*(2^m) using the ITMIA procedures described in the previous Sections. Squaring in *GF*(2^m) is a simple operation, however, as it was stated in (12), we need to perform $m-1$ squarings for m-bit inversion when using the ITMIA procedure of Fig. 2 (similarly, according to (20) we must perform $m-1$ square roots if the algorithm of Fig. 3 is executed). Fortunately, only t field multiplications are needed for computing multiplicative inversion, which is valuable for the design since field multiplication in *GF*(2^m) is a costly and extensive time consuming operation.

Throughout the rest of this Section, we assume that the binary extension field *GF*(2^{193}) was generated using the irreducible trinomial $P(X) = X^{193} + X^{15} + 1$.

Field Squaring Block

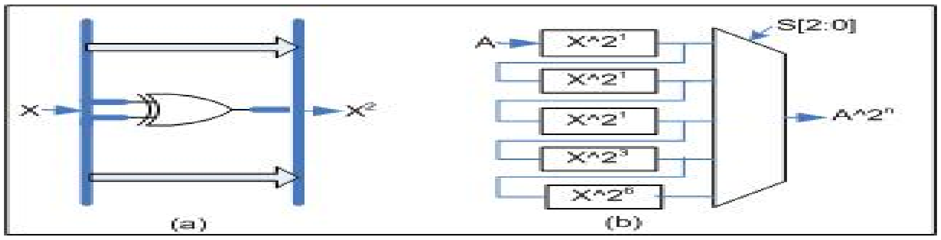

Figure 4: Squarer *GF*(2^{193}) (a) for X^{2^1} (b) for X^{2^n} implementation

Figure 4.a shows our strategy for implementing field squaring trying to use as few clock cycles as possible. In the case of the binary extension field $GF(2^{193})$, field squaring can be obtained using XOR gates only, as it was summarized in (5).

Figure 4.b shows the $GF(2^{193})$ field squarer block used in this work. Referring to the addition chain described in Table 3.3, frequent squaring operations in $GF(2^{193})$ are,

- $[\beta_{u_i}(a)]^{2^1}$ (1 time);
- $[\beta_{u_i}(a)]^{2^3}$ (3 times);
- $[\beta_{u_i}(a)]^{2^6}$ (6 times) and;
- $[\beta_{u_i}(a)]^{2^{12}}$ (12 times).

That is why we took the design decision of cascading 12 field squarer blocks back to back and then, by the appropriate usage of multiplexer blocks, obtain the corresponding outputs after 1, 3, 6, and 12 squarer blocks as shown in Figure 4.b. As an example, the $X^{2^{24}}$ field operation can be accomplished in just two clock cycles by taking the output after the last squarer block (12 squarers) in the first clock cycle and then repeating this operation in a second clock cycle so that we get the required 24 field squarings.

Field Square Root Block

Based on eq. (6), we designed the square root block shown in Figure 5.a. As it can be seen, this field operation can be obtained by using two-input XOR gates only. In a similar fashion to the squaring block discussed above, Figure 5.b implements multiple square root operations trying to save as many clock cycles as possible. The five inputs of the multiplexer are formed by replicating 1, 1, 1, 3, and 6 square root blocks which can perform 1, 2, 3, 6 and 12 square root operations respectively, in just one clock cycle.

Field Multiplier

Our strategy for multiplication is based on the binary Karatsuba-Ofman multiplier which is a variation of normal Karatsuba-Ofman multiplier as it was presented in [13]. Figure 6 shows a $GF(2^{193})$ binary Karatsuba-Ofman multiplier which is a hybrid approach that utilizes Karatsuba-Ofman multiplication with a combination

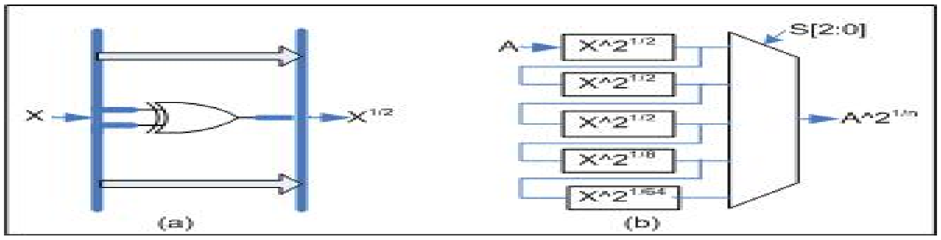

Figure 5: Square Root Circuit $GF(2^{193})$ (a) for $X^{2^{-1}}$ (b) for $X^{2^{-n}}$ implementation

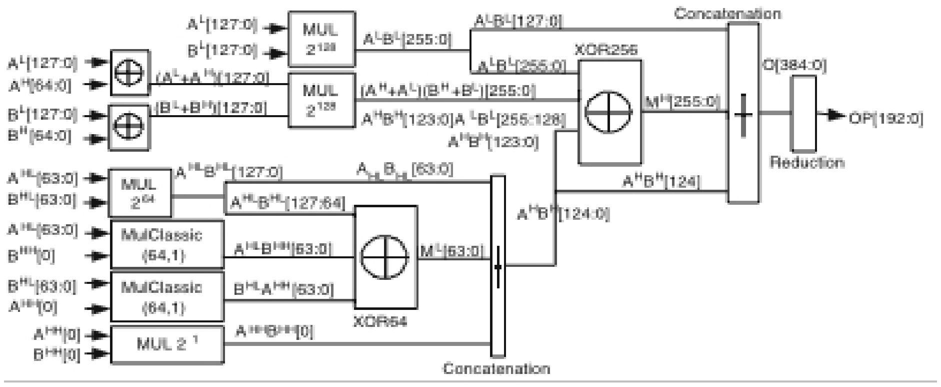

Figure 6: $GF(2^{193})$ binary Karatsuba multiplier

of the classical school-method whenever it is useful. In this design, two 193-bit operands A and B are multiplied by first dividing each operand into two parts: upper part (say A^H and B^H of 128 bits each) and lower part (say A^L and B^L of 65 bits each). For 128-bit multiplications, two Karatsuba-Ofman multipliers were used. However, for 65 bit multiplication, instead of using three 64-bit Karatsuba-Ofman multipliers, only one 64-bit Karatsuba-Ofman multiplier, two $64\times$ 2 classical multiplier (2 multiplexers) and one 1-bit multiplier (only AND gate) were used. Using this approach, savings are made not only in terms of FPGA resources but also in achieving a higher parallelism. Therefore, the time delay of the 193-bit multiplier is equal to the time delay of the biggest multiplier only (time delay of the 128-bit multiplier block).

General Architecture

The proposed architecture for multiplicative inversion includes a square root, squarer and multiplier blocks as shown in Figure 7. Referring to algorithms in Figures 2

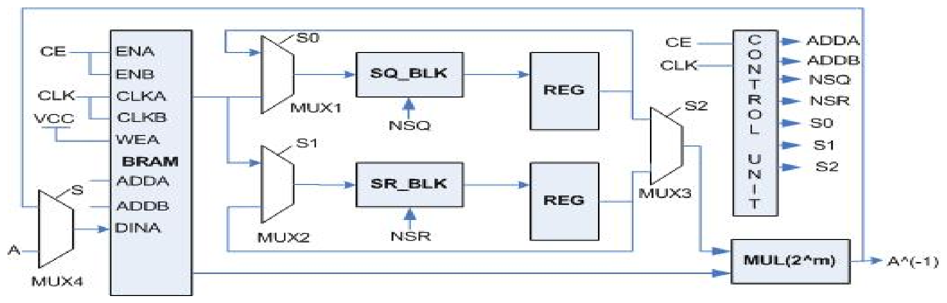

Figure 7: General Architecture for Multiplicative Inversion over $GF(2^{193})$

and 3, squaring-multiplication and square root-multiplication are the two sequences needed for computing multiplicative inverses over $GF(2^{193})$. Both are independent and can be processed in parallel provided that hardware resources meet up design requirements. A direct approach would be to use two multipliers with squarer and square root blocks operating separately. That would, however, be more expensive as our multiplier block consumes a large amount of hardware resources. A more reasonable architecture can be obtained with a single multiplier by introducing a multiplexer for squarer and square-root blocks as shown in Fig. 7. The intermediate results required for next stages of the algorithm are read/written in a Block select RAM (BRAM).

BRAMs are built-in memory modules available in Virtex and VirtexE series devices by Xilinx. A dual port BRAM can be configured into two single port BRAMs, i.e., data can be read/written at two ports simultaneously. This useful feature was exploited in this design in order to achieve higher parallelism. First port was configured as a RAM in order to write the multiplier outputs, while the second one was configured as a ROM, responsible to read the second multiplier operand (already stored from previous iterations). A single BRAM has a size of 4K (4096 bits), which is sufficiently large for storing all intermediate results generated by our algorithm. Additionally, an array of 12 BRAMs was needed for managing a 193-bit data bus.

The inputs/output of the multiplier for writing/reading to/from BRAM are governed by an address scheme. Data paths for squaring, square root and then multiplication are adjusted by providing selection bits for the three multiplexers MUX1, MUX2, and MUX3. MUX4 is used for switching external data during the first cycle and then to feedback data until the final calculation of inversion is obtained. The NSQ and NSR control signals select data path for performing a number of squaring and square root operations. This is done by providing address bits to the multiplexers available inside the SQ_BLK and SR_BLK blocks.

For instance, NSQ=000 selects the first input of the multiplexer which is connected to the output of a single squarer unit, whereas NSQ=101, selects the fifth multiplexer input, which is connected to the 12 squarer unit. A total of 17 bits (4 bits for BRAM port A, 4 bits for BRAM port B, 1 bit for MUX1, 1 bit for MUX2, 1 bit MUX3, 3 bits for NSQ, 3 bits for NSR) are used for controlling and synchronizing the whole circuitry. The 17-bit control word for each clock cycle is filled in the ROM block, and then they are extracted at the rising edge of each clock cycle. A short description of the control unit is given below.

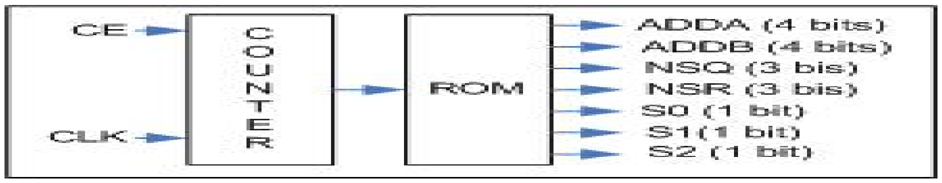

Figure 8: Design Control Unit

As shown in Fig. 7, Control Unit block orchestrates and synchronizes data flow for the whole design. A 4- bit counter and a ROM constitute the control unit. The ROM block is filled with a total of twenty 17-bit control words. Those control words are used at each one of the 20 clock cycles required for completing the execution of our algorithm. The address bits for the ROM block are timely incremented by a 4-bit counter as shown in Figure 8.

Table 3 shows the algorithm dataflow. In the first cycle, the field element a, whose multiplicative inverse is required, is written into the BRAM. Then, starting at cycle 2, our architecture of Fig. 7 computes both of them, $\beta_{u_i}(a)$ and $\gamma_{u_i}(a)$ for $i = 0, 1, \cdots, 7$ in parallel. At clock 21, a final computation, namely, $\gamma_{u_8}(a) = [\beta_{u_7}(a)]^{2^1} \cdot \gamma_{u_7}(a)$, is performed, which according to theorem 4.1 yields the required multiplicative inverse, i.e., $\gamma_{u_8}(a) = a^{-1}$.

Notice that the control word that commands all the operations to be performed in the next rising edge of the master clock, is set at the rising edge of the previous clock cycle. For example at cycle 8, the control word: $0101||0011||000||011$ selects the operations for cycle 9 as follows: address 0101 commands to write data at port A; address 0011 orders to read data from port B; and address 000 commands to perform a single squaring; and finally the address 011 orders to perform six square root operations that will be stored in the register t_2. The code word "DC" denotes the *Don't Care* condition.

Finally, let us say that in the very first computation, the first clock cycle is used for loading the input field element a. If more multiplicative inverse computations are required, the input data can always be loaded during the last clock cycle of the previous computation, thus making possible that a single multiplicative inverse calculation may be accomplished in a total of 20 clock cycles.

Table 3: Algorithm Dataflow

Clock	AdrA‖AdrB‖SQR‖SQROOT	Write Reg1	Write Reg2
1	0000‖0000‖000‖000	Loading input data	
2	0001‖0000‖000‖000	$t_1 = [\beta_{u_0}(a)]^{2^1}$	
3	1001‖0001‖000‖000	$\beta_{u_1}(a) = t_1 \cdot \beta_{u_0}(a)$	$t_2 = [\gamma_{u_0}(a)]^{2^{-1}}$
4	0010‖0000‖000‖000	$t_1 = [\beta_{u_1}(a)]^{2^1}$	$\gamma_{u_1}(a) = t_2 \cdot \gamma_{u_0}(a)$
5	1010‖0000‖010‖000	$\beta_{u_2}(a) = t_1 \cdot \beta_{u_1}(a)$	$t_2 = [\gamma_{u_1}(a)]^{2^{-1}}$
6	0011‖0010‖000‖010	$t_1 = [\beta_{u_2}(a)]^{2^1}$	$\gamma_{u_2}(a) = t_2 \cdot \gamma_{u_1}(a)$
7	1011‖1010‖011‖000	$\beta_{u_3}(a) = t_1 \cdot \beta_{u_2}(a)$	$t_2 = [\gamma_{u_2}(a)]^{2^{-1}}$
8	0101‖0011‖000‖011	$t_1 = [\beta_{u_3}(a)]^{2^6}$	$\gamma_{u_3}(a) = t_2 \cdot \gamma_{u_2}(a)$
9	1100‖1011‖100‖000	$\beta_{u_4} = t_1 \cdot \beta_{u_3}(a)$	$t_2 = [\gamma_{u_3}(a)]^{2^{-6}}$
10	0110‖0101‖000‖100	$t_1 = [\beta_{u_4}(a)]^{2^{12}}$	$\gamma_{u_4} = t_2 \cdot \gamma_{u_3}(a)$
11	1101‖1100‖100‖000	$\beta_{u_5}(a) = t_1 \cdot \beta_{u_4}(a)$	$t_2 = [\gamma_{u_4}(a)]^{2^{-12}}$
12	DC‖ DC ‖100‖100	$t_1 = [\beta_{u_5}(a)]^{2^{12}}$	$\gamma_{u_5}(a) = t_2 \cdot \gamma_{u_4}(a)$
13	0111‖0110‖000‖100	$t_1 = (t_1)^{2^{12}}$	$t_2 = [\gamma_{u_5}(a)]^{2^{-12}}$
14	0111‖1101‖100‖000	$\beta_{u_6}(a) = t_1 \cdot \beta_{u_5}(a)$	$t_2 = (t_2)^{2^{-12}}$
15	DC ‖ DC ‖100‖100	$t_1 = [\beta_{u_6}(a)]^{2^{12}}$	$\gamma_{u_6}(a) = t_2 \cdot \gamma_{u_5}(a)$
16	DC ‖ DC ‖100‖100	$t_1 = (t_1)^{2^{12}}$	$t_2 = [\gamma_{u_6}(a)]^{2^{-12}}$
17	DC ‖ DC ‖000‖100	$t_1 = (t_1)^{2^{12}}$	$t_2 = (t_2)^{2^{-12}}$
18	1000‖0111‖000‖000	$t_1 = (t_1)^{2^{12}}$	$t_2 = (t_2)^{2^{-12}}$
19	1111‖1110‖000‖000	$\beta_{u_7}(a) = t_1 \cdot \beta_{u_6}(a)$	$t_2 = (t_2)^{2^{-12}}$
20	DC ‖1111‖000‖000	$t_1 = [\beta_{u_7}(a)]^{2^1}$	$\gamma_{u_7}(a) = t_2 \cdot \gamma_{u_6}(a)$
21	INV‖INV‖000‖000	$\gamma_{u_8}(a) = t_1 \cdot \gamma_{u_7}(a) = a^{-1}$	

6 Comparison and Results

As it was explained in the previous Section, multiplicative inverse computation over $GF(2^{193})$ was achieved by integrating three main building blocks, namely, squaring, square root and multiplication blocks. Table 4 presents a summary of the implementation results obtained for each individual building block as well as for the whole system, i.e., inversion over $GF(2^{193})$. Xilinx Foundation Tool F4.1i was used for design synthesis, implementation and verification of results. The Binary Karatsuba-Ofman multiplier block occupied 8753 CLB slices executing one field multiplication in 43.1ηS. The field Squarer and square root in $GF(2^{193})$ took a total of 47 and 46 CLB slices for a single block respectively. The architecture was implemented in a XCV3200efg1156 (VirtexE device) occupying a total of 11131 (34.3%) CLB Slices and 12 (5 %) BRAMs. One inversion in $GF(2^{193})$ consumes 0.943μS in 20 clock cycles at a rate of 21.2 MHz (47.16 ηS).

Table 4: Design Implementation Summary

Design	Device (XCV)	CLB slices	Timings
Squarer block $GF(2^{193})$	3200E	47	
Square root block $GF(2^{193})$	3200E	46	
Binary Karatsuba-Ofman Multiplier $GF(2^{193})$	3200E	8753	43.1ηS
Inversion $GF(2^{193})$	3200E 12 BRAMs	11081	0.943μS

Table 5 shows the computational cost of several reported designs for the computation of multiplicative inversion over $GF(2^m)$ in hardware platforms. Furthermore, we show also that an implementation of the *standard* Itoh-Tsujii algorithm using our architecture requires 28 clock cycles, thus computing the multiplicative inverse in about 1.32μS. This implies that the Itoh-Tsujii parallel version proposed in this work represents a saving of about 30% when compared with the standard version.

7 Conclusions

In this paper, a novel derivation of the standard Itoh-Tsujii algorithm that offers a potential speedup when implemented in hardware platforms was presented. At first, we combined the standard Itoh-Tsuii algorithm with the concept of addition chains. Then, we showed that for this version of the Itoh-Tsuii algorithm the

Table 5: Specifications for inversion in *GF*(2^m)

Reference	Field	Cycles	Freq (MHz)	$timings$
Gutub et. al. [6]	*GF*(2^{256})	5000	50	100μS
Goodman et. al. [14]	*GF*(2^{256})	3712	–	–
Bednara et. al. [15]	*GF*(2^{191})	–	50	7.8μS
Lutz [16]	*GF*(2^{163})	259	50	5.18μS
This work (Standard)	*GF*(2^{193})	28	21.2	1.32μS
This work (Parallel)	*GF*(2^{193})	20	21.2	0.943μS

multiplicative inverse of an arbitrary nonzero field element in GF(2^m) can be computed by performing exactly $m-1$ field squarings and t multiplications, where t is the step-length of the optimal addition-chain for m-1.

Furthermore, we derived a novel version of the Itoh-Tsujii algorithm which uses field multiplication, field squaring and field square root operators as main building blocks. We showed how this version of the algorithm can be parallelized when implemented in hardware platforms. Our method achieves its best performance when using a special class of irreducible trinomials, namely, $P(X) = X^m + X^k + 1$, with m and k odd numbers. This is because for this special class of irreducible trinomials, the computation of the field square root operation is simpler than field squaring.

We implemented the proposed algorithm in a reconfigurable hardware device for the computation of multiplicative inverses of nonzero field elements in the finite field *GF*(2^m) generated by the irreducible trinomial $P(X) = X^{193}+X^{15}+1$. Our experimental results show that the parallel version of the Itoh-Tsujii algorithm implementation yields a speedup of about 30% when compared with the standard version of it.

Since for all practical cryptographic and code applications in binary extension fields field multiplication is a mandatory operator, our solution does not represent a significant extra burden in terms of hardware resource requirements.

It is worth to notice that although the theoretical formulae included in this paper were derived assuming polynomial basis representation of the field elements, the extension of our results to optimal normal basis is straightforward.

Future work of this paper includes finding other classes of trinomials were the parallel version of the Itoh-Tsujii algorithm presented here might be useful in terms of performance speedup.

References

[1] T. Itoh and S. Tsujii, “A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis,” *Information and Computing*, vol. 78, pp. 171–177, 1988.

[2] G.L:Feng, “A VLSI architecture for fast inversion in GF(2^m),” *IEEE Transactions on Computers*, vol. 38(10), pp. 1383–1386, October 1989.

[3] N. Takagi, J. Yoshiki, and K. Tagaki, “A fast algorithm for multiplicative inversion in GF(2^m) using normal basis,” *IEEE Transactions on Computers*, vol. 50(5), pp. 394–398, May 2001.

[4] M. A. Hasan, “Efficient computation of multiplicative inverses for cryptographic applications,” in *15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, U.S.A., June 2001.

[5] S. Yen, “Improved normal basis inversion in GF(2^m),” *IEE Electronic Letters*, vol. 33(3), pp. 196–197, January 1997.

[6] A. A.-A. Gutub, A. F. Tenca, E. Savas, and C. K. Koc, “Scalable and unified hardware to compute montgomery inverse in GF(p) and GF(2^n),” *Cryptographic Hardware and Embedded Systems - CHES 2002, 4th International Workshop, Redwood Shores, CA, USA*, vol. 2523, pp. 484–499, August 20002.

[7] F. Rodríguez-Henríquez, N. A. Saqib, and N. Cruz-Cortés, “A fast implementation of multiplicative inversion over GF(2^m),” in *International Symposium on Information Technology (ITCC 2005)*, vol. 1, Las Vegas, Nevada, U.S.A., April 2005, pp. 574–579.

[8] J. Guajardo and C. Paar, “Itoh-tsujii inversion in standard basis and its application in cryptography and codes,” *Designs, Codes and Cryptography*, vol. 25, pp. 207–216, 2002.

[9] D. E. Knuth, *The Art of Computer Programming 3rd. ed.* Reading, Massachusetts: Addison-Wesley, 1997.

[10] I. P1363/D13, *Standard specifications for public-key cryptography*, draft version 13 ed. ”http://grouper.ieee.org/groups/1363/”: IEEE standards documents, November 1999.

[11] K. Fong, D. Hankerson, J. López, and A. Menezes, "Field inversion and point halving revisited," *IEEE Transactions on Computers*, vol. 53, no. 8, pp. 1047–1059, Aug. 2004.

[12] A. J. Menezes, P. C. van Oorschot, and S. A.Vanstone, *Handbook of Applied Cryptography*. Boca Raton, Florida: CRC Press, 1996.

[13] F. Rodríguez-Henríquez, N. A. Saqib, and A. Díaz-Pérez, "A fast parallel implementation of elliptic curve point multiplication over $gf(2^m)$," *Elsevier Journal of Microprocessors and Microsystems*, vol. 28, no. 8, pp. 329–339, Aug. 2004.

[14] J. Goodman and A. P. Chandrakasan, "An energy-efficient reconfigurable public-key cryptography processor," *IEEE Journal of Solid-State Circuits*, vol. 36, no. 11, pp. 1808–1820, Nov. 2001.

[15] M. Bednara, M. Daldrup, J. Shokrollahi, J. Teich, and J. von zur Gathen, "Reconfigurable implementation of elliptic curve crypto algorithms," in *Proc. of The 9th Reconfigurable Architectures Workshop (RAW-02)*, Fort Lauderdale, Florida, U.S.A., April 2002.

[16] J. Lutz, "High performance elliptic curve cryptographic co-processor," Ph.D. dissertation, University of Waterloo, 2003.