Achieving Fast Multiplicative Inversion in GF(2^m)

Francisco Rodríguez-Henríquez, Nazar A. Saqib, and Nareli Cruz-Cortés[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@cs.cinvestav.mx, {nabbas, nareli}@computacion.cs.cinvestav.mx

Abstract. A Fast algorithm for multiplicative inversion in $GF(2^m)$ using addition chains is presented. The algorithm can be seen as an alternative to Itoh-Tsujii method and it has been specially tailored for its efficient implementation in reconfigurable hardware devices. We give all the design details from the algorithm and implementation point of view of a specific design that was realized in a Xilinx device. Our design is able to compute multiplicative inversion in $GF(2^{193})$ in less than 1μS using only 19 clock cycles. Although the design presented here was specifically optimized for the aforementioned finite field, the methodology described in this paper can be used for other finite field sizes.

1 Introduction

Binary extension finite fields $GF(2^m)$ have a paramount importance in many modern applications, such as public and secret key cryptosystems, error-correcting codes (quite particularly, Reed-Solomon codes), etc. In the category of public key cryptosystems, one of the most notorious examples can be found in elliptic Curve Cryptography (ECC), where high performance implementations directly depend on the efficiency in the computation of the underlying finite field arithmetic operations. Although Elliptic curve cryptosystems can be defined also over prime finite fields $GF(P)$, binary finite fields $GF(2^m)$ constitute a more attractive option when a designer is dealing with hardware platforms. This is so because of the inherent simplicity of several arithmetic operations in $GF(2^m)$ where the carry-propagation issue of $GF(P)$ does not exist.

On the other hand, several modern key block cipher algorithms already make use of $GF(2^m)$ finite field arithmetic, being the Advanced Encryption Standard (AES) the most famous example in its category. All the data transformations used in AES are defined over $GF(2^8)$ arithmetic. Moreover, the most important (from the cryptographic point of view) transformation in AES, Byte Substitution, is based on the computation of multiplicative inversion operation over the said binary finite field $GF(2^8)$.

Among the basic arithmetic operations, namely addition, subtraction, multiplication and inversion of nonzero elements, the later is undoubtedly the most time consuming one. The multiplicative inversion of an element $\alpha \in F$ is defined

as the process to find an element $\alpha^{-1} \in F$ such that $\alpha \cdot \alpha^{-1} \equiv 1 \bmod P(x)$. Several algorithms to compute the multiplicative inverse in GF(2^m) have been proposed in literature [1–7]. In [4, 6] the 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 $2m$ cycles [4]. In [7] an architecture to compute the Montgomery multiplicative inverse for both, $GF(P)$ 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]. It was originally proposed to be applied in $GF(2^m)$ using Normal Basis representation. Since its publication however, several improvements and variations of it have been reported [2, 3, 5], showing that it can be used in other field representations too.

Unfortunately enough, one can often detect some resistance from the FPGA design community to use FLT-related techniques for computing multiplicative inverses. This phenomenon is probably due to two 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 can only be applied when using Normal Basis Representation.

In this paper we try to show to the community the advantages of FLT-related techniques by presenting an efficient implementation of a variation of the ITMIA algorithm that uses *addition chains* [8]. We show that this technique yields the most efficient way to compute multiplicative inverses in $GF(2^m)$ using the polynomial (canonical) representation. The design presented here is able to obtain the multiplicative inverse of an nonzero element $\alpha \in GF(2^{191})$ in less than 1μS using a total of 19 clock cycles. The design was implemented in a Xilinx VirtexE FPGA device.

The rest of this paper is organized as follows. In Section 2 some basic finite field concepts are reviewed. In Section 3 we discuss some theoretical aspects of multiplicative inversion operation. In Section 4 the concept of addition chains is discussed in order to introduce the algorithm utilized in this work. Then, in Section 5 all the technical details of the design implemented here are discussed in detail. In Section 6 we summarize the main performance figures of our designs and we also compare with other comparable reported designs. Finally, in Section 7 some conclusions remarks are drawn.

2 Preliminaries: Binary Finite Field Arithmetic

A polynomial P in $GF(q)$ is *irreducible* if P is not a unit element and if $P = FG$ then F or G must be a unit, that is, a constant polynomial.

Let $P(x)$ be an irreducible polynomial over $GF(2)$ of degree m, and let α be a root of $P(x)$, i.e., $P(\alpha) = 0$. Then, $P(x)$ can be used to generate a binary finite field $F = GF(2^m)$ with exactly $q = 2^m$ elements, where α itself is one of those elements. Furthermore, the set $\{1, \alpha, \alpha^2, \ldots, \alpha^{m-1}\}$ forms a basis for F, and is called the polynomial (canonical) basis of the field [9]. Any arbitrary

element $A \in GF(2^m)$ can be expressed in polynomial basis as,

$$A = \sum_{i=0}^{m-1} a_i \alpha^i.$$

In finite fields, all the conventional arithmetic operations (addition, subtraction, multiplication, squaring, exponentiation and multiplicative inversion) are well defined operations, meaning that all those arithmetic operations observe the five Abelian properties for groups.

In binary finite field arithmetic, subtraction and addition are equivalent operations. The addition of two elements $A, B \in F$ is simply defined as the bit-wise XOR addition of two polynomials. Addition is by far the less costly field operation. Thus, its computational complexity is usually neglected.

Let $A(x), B(x), C'(x) \in GF(2^m)$ and $P(x)$ be the irreducible polynomial generating $GF(2^m)$. Then the modular multiplication operation is defined as $C'(x) = A(x)B(x) mod P(x)$. Modular Multiplication might be accomplished in two steps, by performing first a polynomial product of the two operands A and B, followed by a modular reduction step using the generating polynomial $P(x)$. The squaring operation, defined as $A(x)^2 = A(x)A(x) mod P(x)$ is a linear operation in GF(2^m) arithmetic. To see this, let A be an arbitrary element that belongs to the field $GF(2^m)$. Then the square of A can be readily found using the following identity,

$$C(x) = A^2(x) = \sum_{i=0}^{m-1} a_i x^{2i}. \quad (1)$$

Hence, the modular squaring of an element A can be computed by performing two steps: polynomial squaring (achieved by simply interleaving a zero value between each one of the original bits of A), followed by modular reduction by the irreducible polynomial $P(x)$.

Let $A(x) \in GF(2^m)$ and e be an m-bit positive integer, i.e., $m = \lceil \log_2(e) \rceil$. Then the modular exponentiation of an element A raised to the power e is defined as,

$$C(x) = A(x)^e mod P(x). \quad (2)$$

There exist a large number of reported algorithms to efficiently solve Eq. 2 [9, 8], being the binary exponentiation algorithm the most popular for hardware implementations. That is probably because of its simplicity and relatively good performance/cost benefit. The Binary exponentiation algorithm (and in fact most exponentiation algorithms) utilizes modular multiplication and squaring as the most prominent building blocks needed to obtain the desired computation.

3 Multiplicative Inversion over GF(2^m)

Since a quite long time it has been known that Fermat's Little Theorem (FLT) provides a mechanism to compute multiplicative inverses in finite fields [8]. Certainly, FLT establishes that for any nonzero element $\alpha \in GF(2^m)$, the identity

$\alpha^{-1} \equiv \alpha^{2^m-2}$ holds. Therefore, multiplicative inversion can be performed by computing,

$$\alpha^{2^m-2} \;=\; \alpha^{2^1} \times \alpha^{2^1} \times \cdots \alpha^{2^{m-1}} \tag{3}$$

A straightforward implementation of Eq. 3 can be carried out using the binary exponentiation method, which requires $m-1$ field squarings and $m-2$ field multiplications. The ITMIA algorithm on the other hand, reduces the required number of multiplications to $k + hw(m-1) - 2$, where $k \;=\; \lfloor log_2(m-1) \rfloor$ and $hw(m-1)$ are the number of bits and the Hamming weight of the binary representation of $m-1$, respectively. This remarkably saving on the number of multiplications is based on the observation that since $2^m-2 = (2^{m-1}-1)\cdot 2$, then the identity from Fermat's little theorem can be rewritten as $\alpha^{-1} \equiv \alpha^{2^m-2} \equiv \alpha^{(2^{m-1}-1)^2}$. Then ITMIA computes the field element $(2^{m-1}-1)$ using a recursive re-arrangement of the finite field operations. To see how this can be carried out let us first make some definitions.

Definition 1. *Let α be any arbitrary nonzero element in the field $GF(2^m)$. Then we define $\beta_k \in GF(2^m), k$ a positive integer, as*

$$\beta_k \;=\; \alpha^{2^k-1} \tag{4}$$

Lemma 1. *Let k, j be two positive integers. Then, the element $\beta_{k+j} \in GF(2^m)$ can be expressed as,*

$$\beta_{k+j} \;=\; {\beta_k}^{2^j} \cdot \beta_j \;=\; {\beta_j}^{2^k} \cdot \beta_k \tag{5}$$

Proof. Using Definition 1 to substitute β_k and β_j in terms of the variable α we obtain,

$$\begin{aligned} {\beta_k}^{2^j} \cdot \beta_j &= (\alpha^{2^k-1})^{2^j} \cdot \alpha^{2^j-1} \\ &= \alpha^{2^{k+j}-2^j} \cdot \alpha^{2^j-1} \\ &= \alpha^{2^{k+j}-2^j+2^j-1} \\ &= \alpha^{2^{k+j}-1} \;=\; \beta_{k+j} \end{aligned}$$

□

Corollary 1. *Let k be a positive integer. Then*

$$\beta_{2k} \;=\; {\beta_k}^{2^k} \cdot \beta_k \tag{6}$$

Proof. Eq. 6 is a special case of Lemma 3 for $k = j$. □

Theorem 1 (Itoh-Tsujii Algorithm). *Let α be any arbitrary nonzero element in the field $GF(2^m)$. Let us consider the binary expansion of the k-bit positive number $m-1$ and let us assume that the nonzero components of that binary expansion can be listed as,*

$$m-1 = 2^{l_1} + 2^{l_2} + \cdots + 2^{l_{t-1}} + 2^{l_t}.$$

Where $l_1 < l_2 < \cdots < l_{t-1} < l_t = k-1$.

Then the multiplicative inverse of α, *namely* $\alpha^{-1} \in GF(2^m)$, *can be written as,*

$$\begin{aligned}\alpha^{-1} = \alpha^{2^m-2} &= \left(\alpha^{2^{m-1}-1}\right)^2 \\ &= \left[\beta_{2^{l_1}}\left(\beta_{2^{l_2}}\left(\cdots\beta_{2^{l_{t-2}}}\left[\beta_{2^{l_{t-1}}}\left(\beta_{2^{l_t}}\right)^{2^{2^{l_{t-1}}}}\right]^{2^{2^{l_{t-2}}}}\cdots\right)^{2^{2^{l_2}}}\right)^{2^{2^{l_1}}}\right]^2 \end{aligned} \quad (7)$$

Proof. (Sketch) Applying Lemma 3 to the most inner exponentiation of Eq. 7 we get,

$$\beta_{2^{l_{t-1}}}\left(\beta_{2^{l_t}}\right)^{2^{2^{l_{t-1}}}} = \beta_{2^{l_{t-1}}+2^{l_t}}$$

In the same way, Lemma 3 can be utilized recursively to the rest of the exponentiations of Eq. 7. Starting from the most inner exponentiation, we obtain

$$\begin{aligned}\beta_{2^{l_{t-1}}}\left(\beta_{2^{l_t}}\right)^{2^{2^{l_{t-1}}}} &= \beta_{2^{l_{t-1}}+2^{l_t}} \\ \beta_{2^{l_{t-2}}}\left(\beta_{2^{l_{t-1}}+2^{l_t}}\right)^{2^{2^{l_{t-2}}}} &= \beta_{2^{l_{t-2}}+2^{l_{t-1}}+2^{l_t}} \\ &\vdots \\ \beta_{2^{l_1}}\left(\beta_{2^{l_2}+\cdots+2^{l_{t-1}}+2^{l_t}}\right)^{2^{2^{l_1}}} &= \beta_{2^{l_1}+2^{l_2}+\cdots+2^{l_{t-2}}+2^{l_{t-1}}+2^{l_t}} \\ &= \beta_{2^{m-1}} = \alpha^{2^{m-1}-1}\end{aligned}$$

The final squaring step of Eq. 7 yields the required result since,

$\left(\alpha^{2^{m-1}-1}\right)^2 = (\alpha^{2^m-2}) = \alpha^{-1}$ □

Theorem 2. *The overall complexity of Eq. 7 is* $m-1$ *field squarings plus* $N = k + hw(m-1) - 2$ *field multiplications, where* $k = \lfloor log_2(m-1) \rfloor$ *and* $hw(m-1)$ *are the number of bits and the Hamming weight of the binary representation of* $m-1$, *respectively*

Proof. See [1, 3].

In the next section we introduce the concept of an *addition chain* in order to generalize the ITMIA algorithm just discussed.

4 Addition Chains

An *addition chain* U for a positive integer $e = m-1$ of length t is a sequence of positive integers $U = \{u_0, u_1, \cdots, u_t\}$, and an associated sequence of r pairs $V = \{(v_1, v_2 \cdots, v_t\}$ with $v_i = (i_1, i_2), 0 \leq i_2 \leq i_1 < i$, such that:

– $u_0 = 1$ and $u_t = m-1$;
– for each $u_i, 1 \leq i \leq t, u_i = u_{i_1} + u_{i_2}$.

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

$U := 1 \rightarrow 2 \rightarrow 4 \rightarrow 8 \rightarrow 16 \rightarrow 32 \rightarrow 64 \rightarrow 128 \rightarrow 192$

With the associated sequence 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

$U = 1 \rightarrow 2 \rightarrow 3 \rightarrow 6 \rightarrow 12 \rightarrow 24 \rightarrow 48 \rightarrow 96 \rightarrow 192$

Where for all $i \neq 2$ the combinatorial rule is $u_i = u_{i-1} + u_{i-1} = 2u_{i-1}$, whereas for $i = 2$ the rule is given as $u_t = u_{t-1} + u_{t-2}$.

Let l be the shortest length of any valid addition chain for a given positive integer $e = m-1$. Then the problem of finding an addition chain for e with length l is an **NP**-hard problem [9]. The concept of addition chains leads us to a natural way to generalize the Itoh-Tsujii Algorithm. Consider the algo-

Input: An element $\alpha \in GF(2^m)$, an addition chain U of length t for $m-1$ and its associated sequence V.
Output: $\alpha^{-1} \in GF(2^m)$
Procedure MultiplicativeInversion(α, U)
1. $\beta_0 = \alpha$
2. for i from 1 to t do:
3. $\beta_i = (\beta_{i_1})^{2^{u_{i_2}}} \cdot \beta_{i_2}$
4. return $(\beta_t{}^2)$;

Fig. 1. An Algorithm for Multiplicative Inversion using a Generalized Itoh-Tsujii Approach

rithm shown in Fig. 1. That algorithm iteratively computes the β_i coefficients in the exact order stipulated by the addition chain U. Indeed, starting from $\beta_0 = (\alpha)^{2^{u_0}-1} = (\alpha)^{2^{u_0}-1} = \alpha^{2^1-1}$, the algorithm computes the other t β_i coefficients. In the final iteration, after having computed the coefficient $\beta_t = (\alpha)^{2^{m-1}-1}$, the algorithm returns the required multiplicative inversion by performing a regular field squaring, namely, $\beta_t^2 = (\alpha^{2^m-2}) = \alpha^{-1}$.

Example 2. Let us assume that the binary extension field $GF(2^{193})$ has been generated with the trinomial $P(x) = x^{193}+x^{15}+1$, irreducible over $GF(2)$. Let $\alpha \in GF(2^{193})$ be an arbitrary nonzero field element. Then, using the addition chain of Example 1, the algorithm of Fig. 1 will compute the sequence of β coefficients as shown in Table 4. Once again, notice that after having computed the coefficient β_8 the only remaining step is to obtain α^{-1} as, $\alpha^{-1} = \beta_8^2$

Table 1. Algorithm of Fig. 1: β_i Coefficient Generation

i	u_i	rule	$\beta_{i_1}^{2^{u_{i_2}}} \cdot \beta_{i_2}$	$\beta_i = \alpha^{2^u_i - 1}$
0	1	–	–	$\beta_0 = \alpha^{2^1-1}$
1	2	$2u_{i-1}$	$\beta_0^{2^1} \cdot \beta_0$	$\beta_1 = \alpha^{2^2-1}$
2	3	$u_{i-1} + u_{i-2}$	$\beta_1^{2^1} \cdot \beta_0$	$\beta_2 = \alpha^{2^3-1}$
3	6	$2u_{i-1}$	$\beta_2^{2^3} \cdot \beta_2$	$\beta_3 = \alpha^{2^6-1}$
4	12	$2u_{i-1}$	$\beta_3^{2^6} \cdot \beta_3$	$\beta_4 = \alpha^{2^{12}-1}$
5	24	$2u_{i-1}$	$\beta_4^{2^{12}} \cdot \beta_4$	$\beta_5 = \alpha^{2^{24}-1}$
6	48	$2u_{i-1}$	$\beta_5^{2^{24}} \cdot \beta_5$	$\beta_6 = \alpha^{2^{48}-1}$
7	96	$2u_{i-1}$	$\beta_6^{2^{48}} \cdot \beta_6$	$\beta_7 = \alpha^{2^{96}-1}$
8	192	$2u_{i-1}$	$\beta_7^{2^{96}} \cdot \beta_7$	$\beta_8 = \alpha^{2^{192}-1}$

Let us now assess the computational complexity of the algorithm shown in Fig. 1. 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-esime iteration is $u_i - 1$ [10]. 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.

Summarizing, the algorithm of Fig. 1 can find the inverse of any nonzero element of the field using exactly,

$$\begin{aligned} \#Multiplications &= t; \\ \#Squarings &= m - 1. \end{aligned} \tag{8}$$

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

Squarer and Multiplier are the two main building blocks for performing inversion in GF(2^m). Squaring in GF(2^m) is a simple operation however, as it was stated in Eq. 8, it devours $m - 1$ clock cycles performing $m - 1$ squarings for m-bit inversion. However, much less field multiplications are needed for computing the multiplicative inversion, which is valuable for the design since field multiplication in GF(2^m) is a costly and extensive time consuming operation.

Figure 2.a shows our strategy for implementing field squaring trying to use as few clock cycles as possible. Polynomial squaring is free of cost and is obtained by

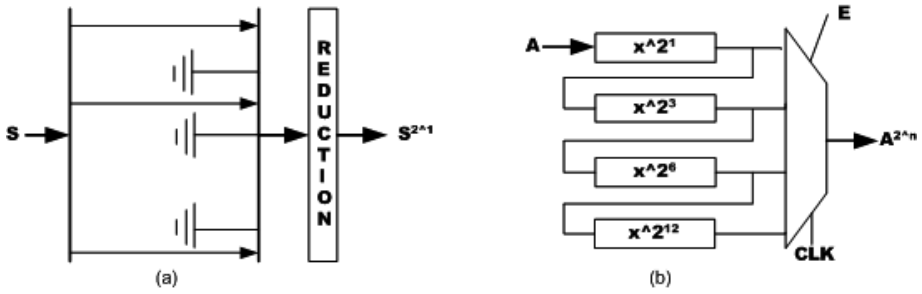

Fig. 2. Squarer GF(2^{193}) (a) for x^{2^1} (b) for x^{2^n} implementation

connecting each alternate even bit to zero (ground). That operation is followed by a second step where reduction is performed by using few XOR gates. Figure 2.b shows the GF(2^{193}) field squarer implementation used in this work. Referring to the addition chain described in Table 4, frequent squaring operations in GF(2^{193}) are ${\beta_i}^{2^1}$ (1 time), $\beta_i 2^3$ (3 times), $\beta_i 2^6$ (6 times), and $\beta_i 2^{12}$ (12 times). That is why we preferred to cascade 12 field squarer blocks back to back and then by the appropriate use of multiplexers obtain the corresponding outputs after 1, 3, 6, and 12 squarer blocks as shown in Figure 2.b. As an example, the $x^{2^{24}}$ field operation can be performed in just two clock cycles by taking the output after the last squarer block (12 squarers) in the first clock cycle and then after a second clock cycle we will get the required 24 field squarings.

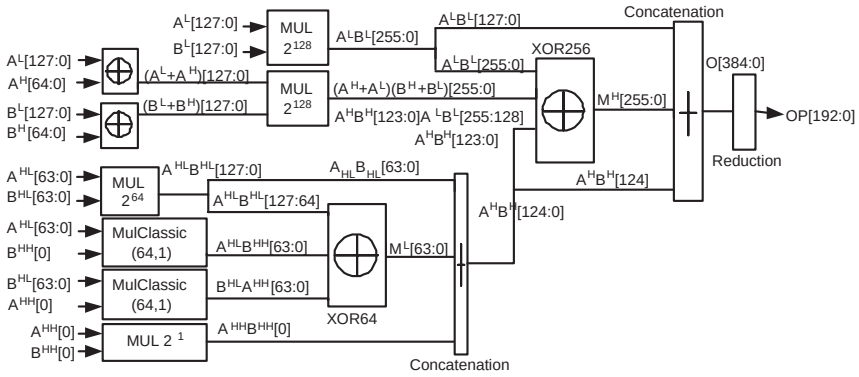

Fig. 3. GF(2^{193}) binary karatsuba multiplier

Our strategy for multiplication is based on the binary Karatsuba-Ofman multiplier which is a variation of normal Karatsuba-Ofman multiplier. Figure 3 shows a GF(2^{193}) binary Karatsuba-Ofman multiplier which is a hybrid approach that utilizes Karatsuba-Ofman multiplication with other multiplication

schemes (classical, etc.) whenever they are useful. In this design example, 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× 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).

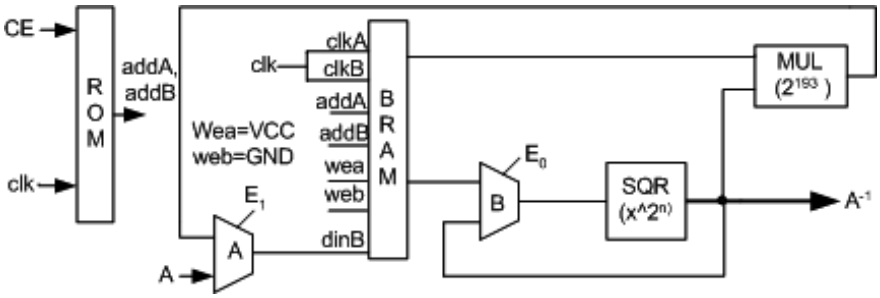

Fig. 4. Inversion in GF(2^{193})

The Squarer block of (Figure 2) and multiplier block of (Figure 3) were the main building blocks used to implement inversion in GF(2^{193}) as shown in Figure 4. In Figure 4, two multiplexers were used. Multiplexer A switches input data in the first cycle and multiplexer B provides a feedback path for squarings until the required number of squarings has been executed. A BRAM (BlockRAM) is used for storing temporary results after multiplications. A BRAM provides fast access to the data and saves a lot of multiplexer operations and registers. One BRAM is sufficient here however the combination of 12 BRAMs provides 193-bit bus length. A dual port BRAM comprises of two independent ports: port A and B. For our architecture, port A and B are always kept in read and write modes respectively. Bit patterns for the required squarings and multiplications were generated according to the inversion algorithm and stored in the ROM (read only memory). Those patterns synchronize the algorithm flow.

Inversion in GF(2^{193}) was implemented on a XCV3200eFG1156 virtexE device using Xilinx Foundation Series F4.1i. It occupied 10065 CLB slices (31%), 12 BRAMs (5%), achieving a frequency of 20.263 MHz (49.350 ηs). The architecture shown in Fig. 4 needs a total of 19 clock cycles to compute the repeated field squarings and the required 10 field multiplications. Thus we are able to perform one multiplicative inversion in GF(2^{193}) in just 0.938 μS.

6 Comparison and Results

The inversion in GF(2^{193}) was made on a XCV3200efg1156 (VirtexE device) using Xilinx Foundation Series F4.1i. Table 2 represents the implementation results for inversion in GF(2^{193}) as well as the building blocks (squarer, multiplier) used for inversion. The Binary Karatsuba-Ofman multiplier GF(2^{193}) block occupied 8753 CLB slices executing one field multiplication in 43.1ηS. The field Squarer GF(2^{193}) took a total of 87 CLB slices. One inversion in GF(2^{193}) consumes 0.938μS by occupying 10065 CLB slices and 12 BRAMs. It took 19 cycles at the rate of 20.26 MHz (49.35ηS).

Table 2. Design Implementation Summary

Design	Device (XCV)	CLB slices	Timings
Squarer GF(2^{193})	3200E	97	
Binary Karatsuba-Ofman Multiplier GF(2^{193})	3200E	8753	43.1ηS
Inversion GF(2^{193})	3200E 12 BRAMs	10065	0.938μS

Table 3 shows the reported computational cost for inversion in GF(2^m). The designs included in this table differ either in the base field or in the implementation platforms. Notice also that the implementations for inversion in GF(2^m) exist using diverse approaches on different platforms but most of them do not report the total time taken to complete one inversion in terms of clock frequency. To the best of our knowledge, our architecture for inversion shows the best performance among the other reported multiplicative inversion implementations using any other approach.

Table 3. Specifications for inversion in GF(2^m)

Reference	Field	Cycles	Frequency (MHz)	*timings*
[11]	GF(2^{256})	5000	50	100μS
[12]	GF(2^{256})	3712	–	–
[13]	GF(2^{191})	–	50	$(7 \times 10 + 190) \times 27.8\eta$S
This work	GF(2^{193})	19	20.26	1.37μS

7 Conclusions

A Fast algorithm for multiplicative inversion in $GF(2^m)$ using addition chains was presented. The algorithm can be seen as an alternative to Itoh-Tsujii method and it has been specially tailored for its efficient implementation in reconfigurable hardware devices. Our design is able to compute multiplicative inversion in $GF(2^{193})$ in less than 1μS using only 19 clock cycles. Although the design presented here was specifically optimized for the aforementioned finite field, the methodology described in this paper can be used for other finite field sizes.

References

1. Itoh, T., Tsujii, S.: A fast algorithm for computing multiplicative inverses in GF(2^m) using normal basis. Information and Computing **78** (1988) 171–177
2. G.L:Feng: A VLSI architecture for fast inversion in GF(2^m). IEEE Transactions on Computers **38(10)** (1989) 1383–1386
3. Takagi, N., Yoshiki, J., Tagaki, K.: A fast algorithm for multiplicative inversion in GF(2^m) using normal basis. IEEE Transactions on Computers **50(5)** (2001) 394–398
4. Hasan, M.A.: Efficient computation of multiplicative inverses for cryptographic applications. In: 15th IEEE Symposium on Computer Arithmetic, Vail, Colorado, U.S.A. (2001)
5. Yen, S.: Improved normal basis inversion in GF(2^m). IEE Electronic Letters **33(3)** (1997) 196–197
6. Hankerson, D., Lopez-Hernandez, J., Menezes, A.: Software implementation of elliptic curve cryptography over binary fields. Cryptographic Hardware and Embedded Systems - CHES 2000, Second International Workshop, Worcester, MA, USA, August 17-18, 2000, Proceedings **1965** (2000) 1–24
7. Gutub, A.A.A., Tenca, A.F., Savas, E., Koc, C.K.: 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 **2523** (20002) 484–499
8. Knuth, D.E.: The Art of Computer Programming 3rd. ed. Addison-Wesley, Reading, Massachusetts (1997)
9. Menezes, A.J., van Oorschot, P.C., A.Vanstone, S.: Handbook of Applied Cryptography. CRC Press, Boca Raton, Florida (1996)
10. von zur Gathen, J., Nöcker, M.: Computing special powers in finite fields: extended abstract. In: Proceedings of the 1999 international symposium on Symbolic and algebraic computation, ACM Press (1999) 83–90
11. Gutub1, A.A.A., Tenca, A.F., Savas, E., Koç, Ç.K.: 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, August 13-15, 2002, Revised Papers **2523** (2003) 485–500
12. Goodman, J., Chandrakasan, A.P.: An energy-efficient reconfigurable public-key cryptography processor. IEEE Journal of Solid-State Circuits **36** (2001) 1808–1820
13. Bednara, M., Daldrup, M., Shokrollahi, J., Teich, J., von zur Gathen, J.: Reconfigurable implementation of elliptic curve crypto algorithms. In: Proc. of The 9th Reconfigurable Architectures Workshop (RAW-02), Fort Lauderdale, Florida, U.S.A. (2002)