

Low-Delay FPGA-Based Implementation of Finite Field Multipliers

José L. Imaña

Abstract—Arithmetic operations over binary extension fields $GF(2^m)$ have many important applications in domains such as cryptography, code theory and digital signal processing. These applications must be fast, so low-delay implementations of arithmetic circuits are required. Among $GF(2^m)$ arithmetic operations, field multiplication is considered the most important one. For hardware implementation of multiplication over binary finite fields, irreducible trinomials and pentanomials are normally used. In this brief, low-delay FPGA-based implementations of bit-parallel $GF(2^m)$ polynomial basis multipliers are presented, where a new multiplier based on irreducible trinomials is given. Several post-place and route implementation results in Xilinx Artix-7 FPGA for different $GF(2^m)$ finite fields are reported. Experimental results show that the proposed multiplier exhibits the best delay, with a delay improvement of up to 4.7%, and the second best $Area \times Time$ complexities when compared with similar multipliers found in the literature.

I. INTRODUCTION

High-speed hardware implementations of arithmetic operations over binary extension fields $GF(2^m)$ are greatly desirable due to their use in cryptography, digital signal processing and code theory [1]. Multiplication is the most complex and important arithmetic operation, because exponentiation, division and inversion can be realized by consecutive use of finite field multiplication [2], [3], [4], [5]. Among the different basis for representation of $GF(2^m)$ elements, polynomial basis (PB) is normally used [6], [7]. In addition to the representation basis, the complexity of the finite field multiplier also depends on the irreducible polynomial $f(y)$ selected for the generation of the binary extension field. In order to perform hardware implementations, low Hamming weight polynomials (trinomials and pentanomials) are used.

$GF(2^m)$ polynomial basis multiplication requires a multiplication of polynomials followed by a modular reduction using an irreducible polynomial [8]. Several bit-parallel PB multipliers in which a *product matrix* combines the above two steps together have been proposed in the literature [9], [10]. In [11], a PB multiplication method based on the decomposition of a product matrix was given. The approach in [11] was applied to five types of irreducible trinomials, and two functions, S_i and T_i , were introduced in such a way that the addition of these functions were used for the computation of the product of two $GF(2^m)$ elements. Functions S_i and T_i are given as addition of products of the coordinates of the two operands. In [12], the additions of products of coordinates included in the S_i and T_i functions obtained for type II irreducible pentanomials were split into sums of 2^k product terms that can be implemented with k -depth binary

trees of XOR gates. It was shown in [12] that the pairwise addition of binary trees with the same depth, given by parentheses used in the expressions of the product coordinates, reduces the theoretical delay of the $GF(2^m)$ multiplier. However, this splitting method enforces strong parenthesized constraints that could hinder a synthesis tool the mapping of S_i and T_i terms into FPGA's logic blocks. A new approach was introduced in [13] for S_i and T_i functions obtained for type II irreducible pentanomials where the splitting is carried out, but the constraint imposed by the pairwise parenthesized addition of binary trees is deleted.

In this brief, low-delay FPGA-based Xilinx implementations of $GF(2^m)$ bit-parallel PB multipliers using irreducible trinomials are given. The novelty of the new multiplier here presented is that the splitting method with the removal of parenthesized constraints has been applied to irreducible trinomials, in such a way that the synthesis tool is free to optimize the implementation. Based on the work in [11], a new general algorithm for the computation of the product coordinates for irreducible trinomials using the new splitting approach without parenthesized constraints is also given. Several finite field multipliers, including SECG [14] recommended fields, have been described in VHDL and the results obtained from their implementations in Xilinx Artix-7 FPGAs have been reported. Experimental post-place and route results show that the new approach for multiplication applied to irreducible trinomials exhibits the best delay, with a delay improvement of up to 4.7%, and the second best $Area \times Time$ complexities when compared with similar bit-parallel polynomial basis multipliers found in the literature.

II. BACKGROUND

Let $f(y) = \sum_{i=0}^m f_i y^i$ be an irreducible polynomial of degree m over the binary field $GF(2)$ and let α be a root of $f(y)$. Any element X of the binary extension field $GF(2^m) = GF(2)[y]/(f(y))$ can be represented in the polynomial basis $\{1, \alpha, \dots, \alpha^{m-1}\}$ as $X = \sum_{i=0}^{m-1} x_i \alpha^i$, with $x_i \in GF(2)$. Two-step classic PB multiplication in $GF(2^m)$ requires a multiplication of polynomials followed by a reduction modulo $f(y)$, although several efficient multiplication methods combining these two steps together by means of a *product matrix* have been proposed [9], [10]. A new $GF(2^m)$ PB multiplication method was given in [11] for the computation of the product $P = X \cdot Z \bmod f(y)$. This method defined functions S_i and T_i given by the addition (XOR) of terms $v_k = (x_k z_k)$ and $w_i^j = (x_i z_j + x_j z_i)$, with $x_i, z_i \in GF(2)$, that can be constructed as binary trees of XOR gates with a bottom level of AND gates (that corresponds with the $x_i z_j$ products). Functions S_i ($1 \leq i \leq m$) and T_i ($0 \leq i \leq m-2$) are given by [12]

$$S_i = x_p + \sum_{h=0}^{p-1} z_h^{i-h-1}, \quad T_i = x_q + \sum_{j=1}^{r-(i+1)} z_{i+j}^{m-j} \quad (1)$$

J.L. Imaña is with the Department of Computer Architecture and Automation, Faculty of Physics, Complutense University, 28040 Madrid, Spain. E-mail: jluimana@ucm.es.

This work has been supported by the Spanish MINECO and CM under grants S2018/TCS-4423 and RTI2018-093684-B-I00.

where $p = \lfloor i/2 \rfloor$ and $q = (\lceil m/2 \rceil + \lfloor i/2 \rfloor)$. In these expressions, $v_p = x_p z_p$ only appears for i odd and v_q only appears for m and i even or for m and i odd, where $r = q$. Otherwise, the term v_q does not occur and $r = (\lceil m/2 \rceil + \lfloor i/2 \rfloor)$. As an example, for $GF(2^7)$ the terms \mathbf{S}_i and \mathbf{T}_i are the following:

$$\begin{aligned} \mathbf{S}_1 &= v_0 = x_0 z_0, \\ \mathbf{S}_2 &= w_0^1 = (x_0 z_1 + x_1 z_0), \\ \mathbf{S}_3 &= v_1 + w_0^2 = x_1 z_1 + (x_0 z_2 + x_2 z_0), \\ \mathbf{S}_4 &= w_0^3 + w_1^2 = (x_0 z_3 + x_3 z_0) + (x_1 z_2 + x_2 z_1), \\ \mathbf{S}_5 &= v_2 + w_0^4 + w_1^3 = x_2 z_2 + (x_0 z_4 + x_4 z_0) + (x_1 z_3 + x_3 z_1), \\ \mathbf{S}_6 &= w_0^5 + w_1^4 + w_2^3 = (x_0 z_5 + x_5 z_0) + (x_1 z_4 + x_4 z_1) + (x_2 z_3 + x_3 z_2), \\ \mathbf{S}_7 &= v_3 + w_0^6 + w_1^5 + w_2^4 = x_3 z_3 + (x_0 z_6 + x_6 z_0) + (x_1 z_5 + x_5 z_1) + (x_2 z_4 + x_4 z_2), \\ \mathbf{T}_0 &= w_1^6 + w_2^5 + w_3^4 = (x_1 z_6 + x_6 z_1) + (x_2 z_5 + x_5 z_2) + (x_3 z_4 + x_4 z_3), \\ \mathbf{T}_1 &= v_4 + w_2^6 + w_3^5 = x_4 z_4 + (x_2 z_6 + x_6 z_2) + (x_3 z_5 + x_5 z_3), \\ \mathbf{T}_2 &= w_3^6 + w_4^5 = (x_3 z_6 + x_6 z_3) + (x_4 z_5 + x_5 z_4), \\ \mathbf{T}_3 &= v_5 + w_4^6 = x_5 z_5 + (x_4 z_6 + x_6 z_4), \\ \mathbf{T}_4 &= w_5^6 = (x_5 z_6 + x_6 z_5), \text{ and} \\ \mathbf{T}_5 &= v_6 = x_6 z_6. \end{aligned}$$

The coordinates of the product $P = X \cdot Z$, with $X, Z \in GF(2^7)$, can be determined by the XOR of these terms.

Irreducible polynomials with low Hamming weight, such as trinomials and pentanomials, are normally used for hardware implementations of $GF(2^m)$ multipliers. Irreducible trinomials [15] $f(y) = y^m + y^n + 1$ are important because they are abundant and, for a given m , an irreducible trinomial can always be found when irreducible pentanomials do not exist. Polynomial basis multiplication using irreducible trinomials was considered in [11], where expressions for the computation of the coefficients of the $GF(2^m)$ product were set in terms of \mathbf{S}_i and \mathbf{T}_i functions. As an example, for the binary field $GF(2^7)$ generated by the trinomial $f(y) = y^7 + y^3 + 1$, the coefficients c_i of the product computed using [11] are given in the second column of Table III. In these expressions, the parentheses point out that the corresponding terms must be shared for different coefficients of the product with the purpose of reducing the area requirements of the multiplier. For example, the addition $(\mathbf{T}_0 + \mathbf{T}_4)$ in Table III appears in c_0 and c_3 , so it can be shared among the two coordinates.

III. DELAY REDUCTION

If the objective is the reduction of the delay, the approach introduced in [11] presents the problem of the unbending construction of \mathbf{S}_i and \mathbf{T}_i terms. For example, for the field $GF(2^7)$ the addition $\mathbf{S}_1 + \mathbf{T}_3 = x_0 z_0 + (x_5 z_5 + (x_4 z_6 + x_6 z_4))$, where the parenthesized terms must be XORed previously to the XOR with the other terms, will result in a binary tree with three levels of XOR gates. However, the addition $\mathbf{S}_1 + \mathbf{T}_3$ requires the sum of the four terms $x_0 z_0$, $x_5 z_5$, $x_4 z_6$ and $x_6 z_4$. If the term $x_0 z_0$ is first added with $x_5 z_5$ and then the sum with $(x_4 z_6 + x_6 z_4)$ is carried out in the form $\mathbf{S}_1 + \mathbf{T}_3 = (x_0 z_0 + x_5 z_5) + (x_4 z_6 + x_6 z_4)$, then a 2-level binary tree of XOR gates can be used for the implementation. This concept was used in [12] in order to describe $GF(2^m)$ PB multipliers based on type II irreducible pentanomials. In [12], functions \mathbf{S}_i and \mathbf{T}_i were split in the form $\mathbf{S}_i = s_k^i \mathbf{S}_k^i + \dots + s_0^i \mathbf{S}_0^i$ and $\mathbf{T}_i = t_k^i \mathbf{T}_k^i + \dots + t_0^i \mathbf{T}_0^i$, where $s_j^i, t_j^i \in GF(2)$ and $k = \lfloor \log_2 m \rfloor$. Terms \mathbf{S}_k^i and \mathbf{T}_k^i represent the sum of 2^j products $x_k z_l$, so they can be implemented using binary trees with j levels of XOR gates. Pairwise addition (XOR) of \mathbf{S}_i^j or \mathbf{T}_i^j terms

TABLE I
FUNCTIONS \mathbf{S}_i AND \mathbf{T}_i FOR $GF(2^7)$.

	2^2	2^1	2^0	binary
\mathbf{S}_1	0	0	$\mathbf{S}_1^0 = v_0$	001
\mathbf{T}_5	0	0	$\mathbf{T}_5^0 = v_6$	
\mathbf{S}_2	0	$\mathbf{S}_2^1 = w_0^1$	0	010
\mathbf{T}_4	0	$\mathbf{T}_4^1 = w_5^6$	0	
\mathbf{S}_3	0	$\mathbf{S}_3^1 = w_0^2$	$\mathbf{S}_3^0 = v_1$	011
\mathbf{T}_3	0	$\mathbf{T}_3^1 = w_4^6$	$\mathbf{T}_3^0 = v_5$	
\mathbf{S}_4	$\mathbf{S}_4^2 = (w_0^3 + w_1^2)$	0	0	100
\mathbf{T}_2	$\mathbf{T}_2^2 = (w_3^6 + w_4^5)$	0	0	
\mathbf{S}_5	$\mathbf{S}_5^2 = (w_0^4 + w_1^3)$	0	$\mathbf{S}_5^0 = v_2$	101
\mathbf{T}_1	$\mathbf{T}_1^2 = (w_2^6 + w_3^5)$	0	$\mathbf{T}_1^0 = v_4$	
\mathbf{S}_6	$\mathbf{S}_6^2 = (w_1^4 + w_2^3)$	$\mathbf{S}_6^1 = w_0^5$	0	110
\mathbf{T}_0	$\mathbf{T}_0^2 = (w_2^5 + w_3^4)$	$\mathbf{T}_0^1 = w_6^6$	0	
\mathbf{S}_7	$\mathbf{S}_7^2 = (w_1^5 + w_2^4)$	$\mathbf{S}_7^1 = w_0^6$	$\mathbf{S}_7^0 = v_3$	111

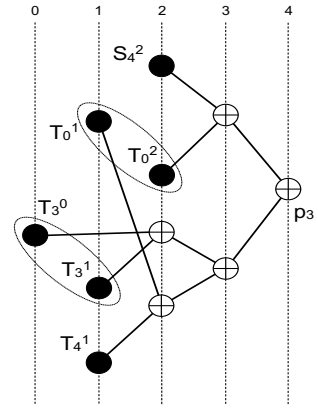


Fig. 1. Implementation of coefficient p_3 for $f(y) = y^7 + y^3 + 1$.

therefore results in a new $(j+1)$ -level binary tree of XOR gates. It can be concluded that if the addition of \mathbf{S}_i^j and \mathbf{T}_i^j functions is done by the pairwise sums of \mathbf{S}_i^j and \mathbf{T}_i^j terms with the same depth j , then the delay of the $GF(2^m)$ multiplier is reduced by means of the diminishing of the XOR levels needed for the implementation.

The above splitting representation introduced in [12] satisfies that the coefficients (s_k^i, \dots, s_0^i) and (t_k^i, \dots, t_0^i) are given by the binary representations of the subindex i for \mathbf{S}_i and of the value $m-1-i$ for \mathbf{T}_i , respectively. In this way, the s_k^i coefficients are '1' if and only if the binary representation of i has a '1' in the position with weight 2^k , and the t_k^i coefficients take the value '1' if and only if the binary representation of $m-1-i$ has a '1' in the position with weight 2^k . For example, the term \mathbf{S}_6 for $GF(2^7)$ is represented by $\mathbf{S}_6 = s_2^6 \mathbf{S}_2^6 + s_1^6 \mathbf{S}_1^6 + s_0^6 \mathbf{S}_0^6 = 1 \cdot \mathbf{S}_2^6 + 1 \cdot \mathbf{S}_1^6 + 0 \cdot \mathbf{S}_0^6 = \mathbf{S}_2^6 + \mathbf{S}_1^6$, where the binary vector $(s_2^6, s_1^6, s_0^6)_2 = (1, 1, 0)_2 = 6_{10}$. Similarly, the term \mathbf{T}_1 for $GF(2^7)$ is given by $\mathbf{T}_1 = t_2^1 \mathbf{T}_2^1 + t_1^1 \mathbf{T}_1^1 + t_0^1 \mathbf{T}_0^1 = 1 \cdot \mathbf{T}_2^1 + 0 \cdot \mathbf{T}_1^1 + 1 \cdot \mathbf{T}_0^1 = \mathbf{T}_2^1 + \mathbf{T}_0^1$, where the binary vector $(t_2^1, t_1^1, t_0^1)_2 = (1, 0, 1)_2$ is the binary representation of the value $(7-1-1)_{10} = 5_{10}$.

Following [12], functions \mathbf{S}_i and \mathbf{T}_i for $GF(2^7)$ are given in Table I, where each function is given as the XOR of the \mathbf{S}_i^j and \mathbf{T}_i^j terms, respectively, existent in their rows. In Table I, terms \mathbf{S}_i and \mathbf{T}_{m-1-i} with the same binary representation are included in a row. Furthermore, the expressions for the corresponding \mathbf{S}_i^j and \mathbf{T}_i^j terms are also given in Table I. For $GF(2^7)$, the coefficients of the product generated by the irreducible trinomial $f(y) = y^7 + y^3 + 1$, with $(m, n) = (7, 3)$, using the algorithms

TABLE II

ALGORITHM FOR MULTIPLICATION FOR $f(y) = y^m + y^n + 1$, WITH
 $1 \leq n \leq (m+1)/2$.

```

if  $n > 1$  then
  for  $i = 0$  to  $n - 2$  do
     $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^i T_i^h + t_h^{i+l} T_{i+l}^h)$ 
  end for
end if
 $p(n-1) = \sum_{h=0}^k (s_h^n S_n^h + t_h^{n-1} T_{n-1}^h)$ 
for  $i = n$  to  $2n - 2$  do
  if  $m$  even and  $n = m/2$  then
     $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^{i-n} T_{i-n}^h)$ 
  else
    if  $i = m - 1$  then
       $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^{i-n} T_{i-n}^h + t_h^{i+l-n} T_{i+l-n}^h)$ 
    else
       $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^i T_i^h + t_h^{i-n} T_{i-n}^h + t_h^{i+l-n} T_{i+l-n}^h)$ 
    end if
  end if
end for
for  $i = 2n - 1$  to  $m - 1$  do
   $j = i$ 
  if  $j \leq m - 2$  then
     $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^j T_j^h + t_h^{i-n} T_{i-n}^h)$ 
  else
     $p(i) = \sum_{h=0}^k (s_h^{i+1} S_{i+1}^h + t_h^{i-n} T_{i-n}^h)$ 
  end if
end for

```

given in [11] and the splitting method given in [12] are given as follows, where parenthesized terms point out that they have to be XORed previously to the XOR with the other terms to reduce the delay of the multiplier:

$$\begin{aligned}
 c_0 &= (S_1^0 + T_0^1) + (T_0^2 + T_4^1), \\
 c_1 &= (S_2^1 + (T_1^0 + T_5^0)) + T_1^2, \\
 c_2 &= (S_3^0 + S_3^1) + T_2^2, \\
 c_3 &= (S_4^2 + T_0^2) + ((T_0^1 + T_4^1) + (T_3^0 + T_3^1)), \\
 c_4 &= (S_5^0 + S_5^2) + (T_1^2 + ((T_1^0 + T_5^0) + T_4^1)), \\
 c_5 &= S_6^2 + (T_2^2 + (S_6^1 + T_5^0)), \text{ and} \\
 c_6 &= (S_7^1 + S_7^2) + ((S_7^0 + T_3^0) + T_3^1).
 \end{aligned}$$

In order to exemplify the approach, the implementation of the coefficient p_3 for the field $GF(2^7)$ is given in Figure 1. This coefficient is the most complex one and can be used to determine the maximum delay of the multiplier for $GF(2^7)$ with $(m, n) = (7, 3)$. In Figure 1, the levels of XOR trees are represented by vertical dashed lines and terms S_i^j and T_i^j are represented by black circles, in such a way that $S_4 = S_4^2$, $T_0 = T_0^2 + T_0^1$, $T_3 = T_3^1 + T_3^0$ and $T_4 = T_4^1$. For example, S_4^2 represents the 2-level binary tree $S_4^2 = (w_0^3 + w_1^2) = (x_0z_3 + x_3z_0) + (x_1z_2 + x_2z_1)$. Furthermore, terms T_0 and T_3 are represented with circles enclosed within ellipses. It can be observed in Figure 1 that the pairwise addition of terms starts with the 0-level term T_3^0 and the 1-level term T_3^1 . In this example, p_3 can be implemented by a binary tree of XOR gates with depth 4, so the delay complexity of the multiplier is $T_A + 4T_X$, with T_A and T_X standing for the delay of 2-input AND and XOR gates, respectively.

IV. LOW-DELAY FPGA-BASED POLYNOMIAL BASIS MULTIPLIER

As previously given, splitting approach applied to $GF(2^m)$ PB multipliers enforces strong parenthesized constraints on the sum of S_i^j and T_i^j terms in order to reduce the number of XOR levels. However these constraints could hinder a synthesis

tool the mapping of these terms into FPGA's configurable logic blocks. If parenthesized constraints are removed, the synthesizer could have more freedom to efficiently implement the $GF(2^m)$ multiplier into the FPGA's logic blocks. This approach was used in [13] for type II irreducible pentanomials where optimized FPGA implementations of bit-parallel multipliers were given. The removal of parenthesized constraints has been applied in this work to irreducible trinomials, obtaining a different set of product equations from those given in [13] due to the different number of existing not-null terms in both polynomials and therefore to the use of different algorithms for the product computation.

In [11], expressions for the coefficients of the $GF(2^m)$ product were given for specific trinomials $f(y) = y^m + y^n + 1$ with $n = m-1, m/2, (m+1)/2, (m-1)/2$ and $n = 1$. However, more general expressions can be given for irreducible trinomials with $1 \leq n \leq (m+1)/2$. Table II shows a new algorithm in which the expressions for the computation of the coefficients of the product given in [11] are generalized for any $1 \leq n \leq (m+1)/2$. In Table II, the new approach without parenthesized constraints is used for the computation of the expressions. This is given by the summation from $h = 0$ to $k = \lfloor \log_2 m \rfloor$ of $s_h^i S_i^h$ and $t_h^i T_i^h$ terms, in such a way that the s_h^i coefficients are '1' if and only if the binary representation of i has a '1' in the position with weight 2^h , and the t_h^i coefficients are '1' if and only if the binary representation of $m-1-i$ has a '1' in the position with weight 2^h . In Table II, the term $l = m-n$ has been used.

Using the new algorithm given in Table II, the expressions for the coefficients of the $GF(2^7)$ PB multiplier based on the irreducible trinomial with $(m, n) = (7, 3)$ are given in the third column of Table III, where the constraints enforced by the parenthesized additions of S_i^j and T_i^j terms have been deleted. In this case, the synthesis tool can be free to better optimize the implementation of the multiplier.

TABLE III

COEFFICIENTS OF THE PRODUCT FOR TRINOMIAL $GF(2^7)$ WITH $n = 3$.

p_0	$S_1 + (T_0 + T_4)$	$S_0^0 + T_2^2 + T_0^1 + T_4^1$
p_1	$S_2 + (T_1 + T_5)$	$S_1^1 + T_1^2 + T_0^1 + T_5^0$
p_2	$S_3 + T_2$	$S_2^2 + S_3^1 + T_2^2$
p_3	$S_4 + (T_0 + T_4) + T_3$	$S_3^3 + T_0^2 + T_0^1 + T_3^1 + T_3^0 + T_4^1$
p_4	$S_5 + (T_1 + T_5) + T_4$	$S_4^4 + S_5^2 + T_2^2 + T_0^1 + T_4^1 + T_5^0$
p_5	$S_6 + T_2 + T_5$	$S_5^5 + S_6^2 + T_2^2 + T_5^0$
p_6	$S_7 + T_3$	$S_6^6 + S_7^1 + S_7^0 + T_3^1 + T_3^0$

The architecture of the new proposed multiplier for irreducible trinomials is shown in Figure 2, where $k = \lfloor \log_2 m \rfloor$. The first (left) block receives the two input operands and generates the $v_i = (x_i z_i)$ functions as the product (AND) of the coordinates of the operands and the $w_i^j = (x_i z_j + x_j z_i)$ functions as the addition (XOR) of coordinate products. The v_i and w_i^j terms implemented in the first block are the inputs to the second block that generates the individual S_i^j and T_i^j small terms as given in Section III. The last block in Figure 2 receives the S_i^j and T_i^j terms and computes the product coordinates as the XOR of these terms using the new multiplication algorithm given in Table II. The main characteristic of this approach in comparison with other multiplication methods is that the coordinates of the product are computed using the individual small terms S_i^j and T_i^j rather than using more complex (parenthesized) expressions, in such a way that a synthesis tool has more freedom to map these terms into FPGA's configurable blocks and therefore optimize the implementation.

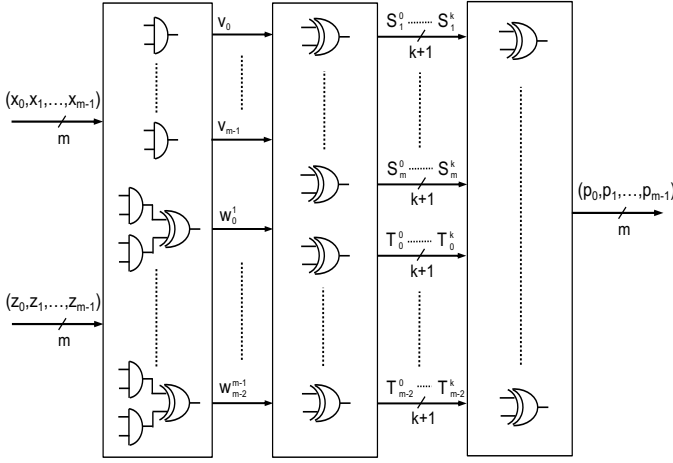


Fig. 2. Block diagram of the proposed multiplier.

TABLE IV

RESULTS OF P&R IMPLEMENTATION FOR SECG TRINOMIALS.

	<i>LUT</i>	<i>SLC</i>	<i>T</i>	<i>SLC</i> × <i>T</i>	<i>LUT</i> × <i>T</i>	<i>DI</i> (%)
$(m, n) = (113, 9)$						
[9]	5429	2847	22.96	65367.12	124649.84	100%
[10]	5394	2714	21.82	59219.48	117697.08	95.0%
[11]	5820	2521	20.21	50949.41	117622.20	88.0%
[6]	6995	3321	21.45	71218.85	150007.78	93.4%
[7]	4453	1845	21.09	38892.60	93869.24	91.9%
[12]	5439	2305	19.79	45615.95	107637.81	86.2%
Fig.2	5443	2303	19.52	44954.56	106247.36	85.0%
$(m, n) = (113, 15)$						
[9]	5430	2866	22.16	63510.56	120328.80	100%
[10]	5397	2706	21.03	56907.18	113498.91	94.9%
[11]	5767	2409	20.37	49071.33	117473.79	91.9%
[6]	6707	3141	20.53	64487.87	137701.42	92.6%
[7]	4460	1874	20.64	38669.99	92032.10	93.1%
[12]	5443	2441	20.22	49357.02	110057.46	91.2%
Fig.2	5431	2226	19.65	43740.90	106719.15	88.7%
$(m, n) = (113, 30)$						
[9]	5424	2793	22.33	62367.69	121117.92	98.0%
[10]	5396	2738	21.30	58319.40	114934.80	93.5%
[11]	5656	2602	20.52	53393.04	116061.12	90.1%
[6]	6088	2792	22.78	63596.18	138672.46	100%
[7]	4432	1860	20.66	38418.30	91542.96	90.7%
[12]	5418	2377	20.53	48799.81	111231.54	90.1%
Fig.2	5429	2361	19.96	47125.56	108362.84	87.6%

V. FPGA IMPLEMENTATION RESULTS

The new proposed polynomial basis multiplier given in Figure 2 that implements the new algorithm given in Table II for irreducible trinomials has been described in VHDL and implemented with Xilinx ISE 14.7 using XST synthesizer in the Artix-7 XC7A200T-FFG1156. In order to compare the new approach with similar PB multipliers found in the literature, the multiplication method given in [10], the bit-parallel version of the multiplier presented in [9] and the multiplier given in [11] that introduced the S_i and T_i functions have been described in VHDL and implemented in Artix-7. Furthermore, recent methods using PB based on the Chinese Remainder Theorem [6] and a combination of Montgomery Multiplication with a Karatsuba-based approach [7] have also been implemented. Finally, the splitting method with strong parenthesized constraints given in [12] applied to irreducible trinomials has also been implemented in order to compare their results with those obtained by the new non-restricted approach. Several finite fields have been implemented in all cases

TABLE V

RESULTS OF P&R IMPLEMENTATION OF $GF(2^{124})$ MULTIPLIERS.

	<i>LUT</i>	<i>SLC</i>	<i>T</i>	<i>SLC</i> × <i>T</i>	<i>LUT</i> × <i>T</i>	<i>DI</i> (%)
$(m, n) = (124, 19)$						
[9]	6522	3122	24.25	75708.50	158158.50	100%
[10]	6494	2205	20.20	44541.00	131178.80	83.3%
[11]	6925	2404	21.18	50916.72	146671.50	87.3%
[6]	8014	2905	20.18	58622.90	161722.52	83.2%
[7]	5614	2357	21.69	51123.33	121767.66	89.4%
[12]	6543	2353	20.79	48918.87	136028.97	85.7%
Fig.2	6529	2052	19.94	40916.88	130188.26	82.2%
$(m, n) = (124, 37)$						
[9]	6509	3115	23.86	74323.90	155304.74	100%
[10]	6497	2194	20.31	44560.14	131954.07	85.1%
[11]	6708	2216	21.30	47200.80	142880.40	89.3%
[6]	7187	2565	20.96	53800.88	150747.33	87.8%
[7]	5507	2330	21.31	49649.97	117348.66	89.3%
[12]	6560	2008	21.62	43412.96	141827.20	90.6%
Fig.2	6501	1971	19.85	39124.35	129044.85	83.2%
$(m, n) = (124, 45)$						
[9]	6512	3126	22.88	71522.88	148994.56	100%
[10]	6497	2182	21.66	47262.12	140725.02	94.7%
[11]	6730	2287	21.70	49627.90	146041.00	94.8%
[6]	6811	2518	22.04	55501.76	150128.06	96.3%
[7]	5495	2450	22.63	55431.25	124324.38	98.9%
[12]	6570	2131	20.31	43280.61	133436.70	88.8%
Fig.2	6526	2016	20.35	41025.60	132804.10	88.9%
$(m, n) = (124, 55)$						
[9]	6510	3132	24.95	78143.40	162424.50	100%
[10]	6495	2046	20.21	41349.66	131263.95	81.0%
[11]	6656	2187	20.45	44724.15	136115.20	82.0%
[6]	6488	2369	21.63	51234.36	140315.98	86.7%
[7]	5486	2381	21.66	51579.60	118843.22	86.8%
[12]	6590	2055	21.15	43463.25	139378.50	84.8%
Fig.2	6513	2086	19.88	41469.68	129478.44	79.7%

using *speed high* optimizations and same pin assignments.

Table IV shows experimental post-place and route results for the binary fields $GF(2^{113})$ recommended by SECG (Standards for Efficient Cryptography Group) [14] with $n = 9, 15$ and 30 . Table V includes results for $GF(2^m)$ multipliers using irreducible trinomials with values $(m, n) = (124, 19), (124, 37), (124, 45), (124, 55)$, and Table VI shows results for $(162, 27), (162, 63)$ and $(162, 81)$. Area complexity is given by the number of *LUTs* and *Slices* (*SLC*) used, and time results *T* (in nanoseconds) correspond with the critical path delay of the multipliers. The *Area* × *Time* metrics are given by the products *SLC* × *T* and *LUT* × *T* in order to compare the area and delay (less is better). Furthermore, *DI*(%) represents the delay improvement with respect to the worst time delay (100%).

From the experimental results, it can be observed that the new multiplier exhibits the lowest delay in all cases except for the multiplier $(m, n) = (124, 45)$, where the delay of the proposed multiplier is only 0.2% higher than the approach given in [12] using splitting method with hard parenthesized constraints. In the remaining implementations, it must be noted that the delay improvement of the new multiplier with respect to the second best time delays ranges from 1.3%, for the multiplier $(124, 19)$ in [10], to 4.7%, for $(162, 27)$ in [12]. Furthermore, the delay improvement with respect to the worst time delay in [9] ranges from 11.2% for $(124, 45)$ to 20.3% for $(124, 55)$. With respect to the area complexity, the multiplier given in [7] presents the lowest number of *LUTs* in all cases, and the lowest number of slices in five out of the ten implemented multipliers (for $GF(2^{113})$, $(162, 27)$ and $(162, 81)$). In four of the remaining cases, the multiplier here presented exhibits the lowest number of slices. With regard to

TABLE VI
RESULTS OF P&R IMPLEMENTATION OF $GF(2^{162})$ MULTIPLIERS.

	LUT	SLC	T	$SLC \times T$	$LUT \times T$	$DI(\%)$
$(m, n) = (162, 27)$						
[9]	11089	4491	24.27	108996.57	269130.03	100%
[10]	11082	3683	23.81	87692.23	263862.42	98.1%
[11]	11657	3869	21.89	84692.41	255171.73	90.2%
[6]	13498	4856	22.22	107880.90	299871.57	91.6%
[7]	8893	3027	23.15	70059.92	205828.49	95.4%
[12]	11101	3925	21.66	85015.50	240447.66	89.2%
Fig.2	11085	3468	20.64	71579.52	228794.40	85.0%
$(m, n) = (162, 63)$						
[9]	11061	4050	23.91	96835.50	264468.51	100%
[10]	11101	3666	21.65	79368.90	240336.65	90.5%
[11]	11281	3704	21.50	79636.00	242541.50	89.9%
[6]	11344	3987	22.28	88818.40	252710.29	93.2%
[7]	9125	3420	22.16	75800.88	202246.50	92.7%
[12]	11137	3741	21.67	81067.47	241338.79	90.6%
Fig.2	11056	3397	20.99	71303.03	232065.44	87.8%
$(m, n) = (162, 81)$						
[9]	11070	3657	23.02	84184.14	254831.40	97.1%
[10]	11108	3694	21.30	78682.20	236600.40	89.9%
[11]	11800	3922	22.65	88833.30	267270.00	95.6%
[6]	10777	3831	23.01	88139.82	247946.44	97.1%
[7]	8897	3428	23.70	81226.46	210814.42	100%
[12]	11065	3501	21.12	73941.12	233692.80	89.1%
Fig.2	11058	3623	20.74	75141.02	229342.92	87.5%

the $Area \times Time$ metrics, the multipliers given in [7] exhibits the lowest $LUT \times T$ values and the lowest $SLC \times T$ values in five out of the ten multipliers. For all the $GF(2^{124})$ multipliers and for (162,63), the multiplier here proposed presents the lowest $SLC \times T$ values, with an improvement with respect to the second best $SLC \times T$ values ranging from 5.2% in (124,45) to 9.9% for (124,37). It is important to note that the work in [7] performs the Montgomery $GF(2^m)$ polynomial basis multiplication given by $P = X \cdot Z \cdot y^h \bmod f(y)$, with $1 \leq h \leq m$, that is different from the multiplication $P = X \cdot Z \bmod f(y)$ here presented. If we do not consider the Montgomery multiplier given in [7], then the proposed multiplier presents the lowest $SLC \times T$ values except for (124,55) and (162,81), with improvements with respect to the second best values ranging from 1.4% for (113,9) to 15.5% in (162,27). With regard to $LUT \times T$ values, the proposed multiplier presents the best values in all cases, with improvements ranging from 0.5% for (124,45) to 4.8% in (162,27).

It can also be observed that experimental results show that the multiplier here presented that applies the new non-restricted splitting approach to irreducible trinomials exhibits the lowest delay and $Area \times Time$ values in comparison with the splitting method with hard parenthesized constraints given in [12]. This is because in the parenthesized implementation the synthesizer can not optimize the mapping into the FPGA's configurable logic blocks due to the constraints imposed by the parenthesis. This optimization can be performed in the non-parenthesized version here presented that gives the synthesizer more freedom to find efficient implementations of $GF(2^m)$ bit-parallel PB multipliers.

VI. CONCLUSION

In this brief, low-delay FPGA-based implementations of $GF(2^m)$ bit-parallel polynomial basis multipliers using irreducible trinomials have been given. The novelty of the new multiplier here presented is that the splitting method without parenthesized constraints has been applied to irreducible trinomials, in such a way that the synthesis tool is free to optimize

the implementation. The architecture of the new proposed multiplier and a new general algorithm for the computation of the product for irreducible trinomials $f(y) = y^m + y^n + 1$ for any $1 \leq n \leq (m+1)/2$ using the new splitting approach without parenthesized constraints have also been given. Several $GF(2^m)$ polynomial basis multiplication methods have been described in VHDL and implemented for several field sizes, including SECG recommended fields. Post-place and route implementation results in Xilinx Artix-7 have also been reported. Experimental results have shown that the multiplier here presented exhibits the best delay, with a delay improvement of up to 4.7%, and the second best $Area \times Time$ complexities when compared with similar multipliers found in the literature. It has also been observed that the new proposed multiplier that applies the non-restricted splitting approach to irreducible trinomials exhibits the lowest delay and $Area \times Time$ values in comparison with the splitting method with hard parenthesized constraints. This is because in the parenthesized implementation the synthesizer can not optimize the mapping into the FPGA's configurable logic blocks due to the constraints imposed by the parenthesis. This optimization can be performed in the new non-parenthesized version here presented that gives the synthesizer more freedom to find efficient implementations of $GF(2^m)$ bit-parallel polynomial basis multipliers.

REFERENCES

- [1] A.A.H. Abd-Elkader, M. Rashdan, E.S.A.M. Hasaneen and H.F.A. Hamed, 'FPGA-Based Optimized Design of Montgomery Modular Multiplier', *IEEE Trans. Circuits and Systems II-Express Briefs*, early access, 2021.
- [2] P. H. Namin, R. Muscedere and M. Ahmadi, 'A Fully Serial-In Parallel-Out Digit-level Finite Field Multiplier in \mathbb{F}_{2^m} Using Redundant Representation', *IEEE Trans. Circuits and Systems II-Express Briefs*, vol. 64, no. 11, pp. 1337-1341, November 2017.
- [3] Y. Li, Y. Zhang, X. Guo and C. Qi, 'N-Term Karatsuba Algorithm and Its Application to Multiplier Designs for Special Trinomials', *IEEE Access*, vol.6, pp. 43056-43069, Aug. 2018.
- [4] S.-M. Park, K.-Y. Chang, D. Hong and C. Seo, 'Low Space Complexity $GF(2^m)$ Multiplier for Trinomials Using n-Term Karatsuba Algorithm', *IEEE Access*, vol.7, pp. 27047-27064, March 2019.
- [5] S.-M. Park, K.-Y. Chang, D. Hong and C. Seo, 'Efficient Bit-Parallel Multiplier for All Trinomials Based on n-Term Karatsuba Algorithm', *IEEE Access*, vol.8, pp. 173491-173507, Oct. 2020.
- [6] H. Fan, 'A Chinese remainder Theorem Approach to Bit-Parallel $GF(2^m)$ Polynomial Basis Multipliers for Irreducible Trinomials', *IEEE Trans. Comput.*, vol.65, no.2, pp. 343-352, Feb. 2016.
- [7] Y. Li and Y. Chen, 'New bit-parallel Montgomery multiplier for trinomials using squaring operation', *INTEGRATION, the VLSI journal*, vol.52, pp. 142-155, Jan. 2016.
- [8] C. Paar, 'Efficient VLSI Architectures for Bit Parallel Computation in Galois Fields', PhD Thesis, Universität GH Essen, 1994.
- [9] B. Rashidi, R.R. Farashahi and S.M. Seyedi, 'Efficient Implementation of Low Time Complexity and Pipelined Bit-Parallel Polynomial Basis Multiplier over Binary Finite Fields', *Int. Journal of Information Security*, vol. 7, no. 2, pp. 101-114, July 2015.
- [10] A. Reyhani-Masoleh and M.A. Hasan, 'Low Complexity Bit Parallel Architectures for Polynomial Basis Multiplication over $GF(2^m)$ ', *IEEE Trans. Computers*, vol. 53, no. 8, pp. 945-959, August 2004.
- [11] J.L. Imaña, J.M. Sánchez and F. Tirado, 'Bit-Parallel Finite Field Multipliers for Irreducible Trinomials', *IEEE Trans. Computers*, vol. 55, no. 5, pp. 520-533, May 2006.
- [12] J.L. Imaña, 'High-Speed Polynomial Basis Multipliers over $GF(2^m)$ for Special Pentanomials', *IEEE Trans. Circuits and Systems I-Regular Papers*, vol. 63, no. 1, pp. 58-69, January 2016.
- [13] J.L. Imaña, 'Reconfigurable implementation of $GF(2^m)$ bit-parallel multipliers', *Design, Automation & Test in Europe Conference & Exhibition DATE 2018*, pp. 899-902, March 2018.
- [14] SEC 2. Standards for Efficient Cryptography Group, 'Recommended Elliptic Curve Domain Parameters'. Version 1.0, 2000.
- [15] H. Fan and Y. Dai, 'Fast Bit Parallel $GF(2^m)$ Multiplier for All Trinomials', *IEEE Trans. Comput.*, vol.54, no.4, pp. 485-490, Apr. 2005.