An Optimized M-Term Karatsuba-Like Binary Polynomial Multiplier for Finite Field Arithmetic

Madhan Thirumoorthi, *Graduate Student Member, IEEE*, Moslem Heidarpur, *Member, IEEE*, Mitra Mirhassani, *Senior Member, IEEE*, and Mohammed Khalid, *Senior Member, IEEE*

***Abstract*— Finite field multiplication is a fundamental and frequently used operation in various cryptographic circuits and systems. Because of its high complexity, this operation generally determines the overall complexity and cost of these systems. Therefore, finite field multipliers and their hardware implementation have received considerable attention from researchers. This article proposes a methodology to design an efficient Galois field multiplier. First, space and time complexities for theoretical and field-programmable gate array (FPGA) implementations of M-term Karatsuba-like finite field multipliers were obtained. In addition, an algorithm was developed to obtain an efficient design based on a composite M-term Karatsuba-like multiplier. Furthermore, the proposed multipliers were verified and implemented on various FPGA devices, and implementation results were presented. Reported device utilization and latency indicated that the proposed multiplier is roughly 26% faster and 15% more efficient in the area–delay product compared to the standard Karatsuba multiplier. Moreover, comparison with state of the art also indicated that the proposed design is leading in terms of effectiveness and speed.**

***Index Terms*— Binary polynomial multiplier, field-programmable gate array (FPGA), finite field multiplication, M-term Karatsuba-like.**

I. Introduction

With the ever-growing expansion of modern information technologies in almost every field, the number of threats and importance of information security are increasing day by day. Cryptography systems play a crucial role in ensuring safety and security of information [1]–[5]. In these systems, a fundamental and frequently used operation that determines the overall speed and cost of systems is finite field multiplication. Therefore, the efficiency of the multiplier is of paramount importance [6]–[14].

Among various polynomial multiplication algorithms, school-book multiplication (SBM) is the simplest form of multiplication. For two polynomials of $n-1$ degree, the SBM has complexity of $O(n^2)$. In order to improve the efficiency of multiplication, several algorithms have been proposed by researchers [15]–[25]. One widely known algorithm is the Karatsuba–Ofman multiplier (KOM) [26]. It is a recursive multiplicative approach that has a lower space complexity $[O(n^{\log_2 3})]$ compared to conventional SBM [1]. This approach splits the operands into lower and upper parts, and uses three submultipliers to compute the product. Although KOM can reduce resource requirements, it has the disadvantage of higher delay compared to SBM because of its submultiplier-based recursive structure [27]. To overcome this problem, a series of Karatsuba modifications and various implementation strategies were proposed.

Overlap-free Karatsuba [28] was proposed to eliminate the higher combinational delay of general Karatsuba. A low-complexity Karatsuba multiplier [29] introduced a new implementation strategy to eliminate the high register complexity in current systolic implementation, which leads to an increase in area and power consumption. Samanta *et al.* [30] presented a modified Karatsuba implementation for 8-bit operands, in which terms are separated into different formats to reduce operational latency. Li *et al.* [31] proposed a new type of non-recursive Mastrovito multiplier for GF(2^m) using an n-term Karatsuba algorithm (KA). In Chiou-Yng *et al.*'s work [17], they presented an efficient digit-level parallel-in–serial-out (PISO) multiplier with subquadratic space complexity using the overlap-free Karatsuba multiplication algorithm. Some of the relevant studies are [19] and [32]–[35].

The M-term Karatsuba-like approach has received much attention in recent years. In these Karatsuba-like algorithms, the operand can be divided into a higher number of terms compared to only two terms in Karatsuba. This allows reducing the number of recurrent operations in the KOA and, hence, increasing the speed of multiplication. In the M-term Karatsuba-like multiplier, the more the number of terms, the larger the number of submultipliers is needed.

Montgomery [36] introduced five-, six-, and seven-term Karatsuba-like algorithms that split each polynomial into five, six, and seven parts, and uses recursive construction to perform the multiplication. Based on Montgomery's work, Fan *et al.* [28] proposed a method to obtain more Karatsuba-like formulas by generalizing the division algorithm. Find and Peralta [37] gave a detailed account of M-term Karatsuba-like, where $M = 4, 5, 6$, and 7, and their theoretical representation. Compared to previous works, they achieved smaller size and depth by optimizing the existing M-term Karatsuba-like algorithm.

Manuscript received September 27, 2021; revised December 18, 2021; accepted January 25, 2022. Date of publication February 14, 2022; date of current version April 26, 2022. This work was supported in part by FedDev Ontario and in part by Windsor-Essex Economic Development Corporation (WE EDC). *(Corresponding author: Madhan Thirumoorthi.)*

The authors are with the Department of Electrical and Computer Engineering, University of Windsor, Windsor, ON N9B 3P4, Canada (e-mail: thirumo@uwindsor.ca).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TVLSI.2022.3148207.

Digital Object Identifier 10.1109/TVLSI.2022.3148207

It has been theoretically proven that M-term Karatsuba multipliers, where $M = 3, 4, 5, 6$, and 7, have better performance for large degree polynomials since it reduces the number of recurrence stages and can be recommended as a viable alternative to the regular KOM [36], [37]. However, these Karatsuba-like formulas contain combinations of the submultipliers, and the total delay and area are not linear, especially when physically implemented on the hardware. To optimize the combinational delay of these Karatsuba-like multipliers, a new methodology is proposed and verified with field-programmable gate array (FPGA) implementation.

The main contributions of this article are given as follows.

1) We investigated the gate-level space and time complexity analyses for a larger number of terms, such as five-, six-, and seven-term Karatsuba-like algorithms.
2) We designed a road map to achieve an efficient finite field multiplier using the M-term Karatsuba-like algorithm.
3) We experimentally evaluated the proposed hardware on FPGA where the result indicated the average of 26% delay reduction and 15% lower area–delay product compared to standard KOM.

In this article, first, the performance of M-term Karatsuba-like polynomial multipliers is evaluated both theoretically and based on implementation results. Furthermore, the hardware design space is explored for the various operand sizes in terms of area [number of look up tables (LUTs)], delay, and ADP. We have six degrees of freedom to select the most efficient M-term Karatsuba-like multiplier for each operand size in this step. Second, an improved composite model based on M-term Karatsuba-like and SBM is proposed.

This composite solution uses the M-term Karatsuba-like at the higher level of recurrence and utilizes single-step conventional SBM in a lower recurrence stage. The design has three degrees of freedom for selecting the most efficient composite model. Overall, there are 18 degrees of freedom in the proposed road map to select the number of terms in M-term Karatsuba-like and combination levels. The proposed design was first validated against Python results, described in VHDL, synthesized, and implemented on various FPGA devices, and furthermore, performance metrics and implementation costs were determined.

The proposed multiplier demonstrated roughly 26% reduction in combinational delay and collectively has 15% lower ADP compared to the standard KOM method. Comparison with state of the art also indicated the effectiveness of the design.

The rest of this article is organized as follows. The generalization of M-term Karatsuba-like algorithms is presented in Section II. The implementation of the M-term Karatsuba-like model on FPGAs has been introduced in Section III. Discussions and significance of the proposed method are given in Section IV. Finally, Section V concludes this article.

II. Background

In the finite field arithmetic, multiplication requires a binary polynomial multiplier followed by the modular reduction operation with an irreducible polynomial. The multiplicative cost and additive cost determine the space complexity of implementing this multiplier. The number of combinational gates required to implement the multiplier is used to assess space complexity, while delay complexity is determined by the linear addition of standard gate delays. In this section, the space and delay complexities of the SBM and M-term Karatsuba-like are assessed.

A. SBM Algorithm

Considering A and B as two degree one binomial, the polynomial multiplier using SBM could be realized as

$$A(x) = A_1 x + A_0 \tag{1}$$

$$B(x) = B_1 x + B_0 \tag{2}$$

where A and B is split into two parts of (A_0, A_1) and (B_0, B_1)

$$A(x).B(x) = A_1.B_1 x^2 + (A_1.B_0 + A_0.B_1)x + A_0.B_0. \tag{3}$$

The polynomial multiplication of two degrees one binomial could be calculated, as described in trinomial (3), by employing four point multiplications and three point additions. The theoretical complexities for hardware implementation of this multiplier are given by the number of AND and XOR gates utilized. The maximum delay is the total number of gates in the critical path of the multiplier. The theoretical complexities are given as follows:

$$S_{\text{XOR}} = (n-1)^2 \tag{4}$$

$$S_{\text{AND}} = (n)^2 \tag{5}$$

$$S_{\text{Total}} = S_{\text{XOR}} + S_{\text{AND}} \tag{6}$$

$$T_{\text{Total}} = T_{\text{AND}} + T_{\text{XOR}}. \tag{7}$$

S_{XOR} (4) and S_{AND} (5) gives the space complexity in terms of number of XOR and AND gates for operand size of n. S_{Total} (6) provides overall space complexity of the SBM multiplier. T_{XOR} and T_{AND} denote the linear addition of the combinational delay of XOR and AND gates. T_{Total} (7) represents the total delay in the critical path of the multiplier.

B. M-term Karatsuba-Like Multipliers

An M-term Karatsuba-like multiplier breaks the operands into smaller size operands and uses a number of submultipliers to recursively calculate the product. M-term Karatsuba uses a similar concept as KOM but splits to a higher number of equal parts. For the rest of this article, we assume that each operand is split into M number of polynomials with equal length.

Table I presents recursive products and reconstruction steps required for M-term Karatsuba-like multipliers from $M = 2$ to $M = 7$ for different operand sizes (n). Here, for simplicity, we chose n and M to be equal. Each operand polynomials are first split into M equal parts denoted as $A_0, A_1, \ldots, A_M$ and $B_0, B_1, \ldots, B_M$. Partial products are represented as $P_0, P_1, \ldots, P_k$, where "k" is the number of partial products. The recursive products and reconstruction steps required for each M-term are tabulated in this table. In each recursion, the

TABLE I

GENERALIZATION OF M-TERM KARATSUBA-LIKE FUNCTIONS IN TERMS OF RECURSIVE PRODUCT AND RECONSTRUCTION. A AND B ARE THE TWO OPERANDS (POLYNOMIALS) OF THE MULTIPLICATION IN EACH M-TERM METHOD REPRESENTED IN THIS RESEARCH. $P0, P1, \ldots, P_n$ REPRESENT PARTIAL PRODUCTS OF SUBMULTIPLIERS, AND $R0, R1, \ldots, R_n$ ARE USED TO CONSTRUCT THE FINAL PRODUCT

M , n	Recursive Product	Reconstruction
2	P0 = A_0B_0, P1 = $(A_0 + B_0)(A_1 + B_1)$, P2 = A_1B_1.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P2.
3	P0 = A_0B_0, P1 = A_1B_1, P2 = $(A_0 + A_1)(B_0 + B_1)$, P3 = A_2B_2, P4 = $(A_0 + A_2)(B_0 + B_2)$, P5 = $(A_1 + A_2)(B_1 + B_2)$.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P4, R3 = P1 $\oplus$ P3 $\oplus$ P5, R4 = P0 $\oplus$ P3.
4	P0 = A_0B_0, P1 = A_1B_1, P2 = $(A_0 + A_1)(B_0 + B_1)$, P3 = A_2B_2, P4 = $(A_0 + A_2)(B_0 + B_2)$, P5 = A_3B_3, P6 = $(A_1 + A_3)(B_1 + B_3)$, P7 = $(A_2 + A_3)(B_2 + B_3)$, P8 = $(A_0 + A_1 + A_2 + A_3)(B_0 + B_1 + B_2 + B_3)$.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P4, R3 = P0 $\oplus$ P1 $\oplus$ P2 $\oplus$ P3 $\oplus$ P4 $\oplus$ P5 $\oplus$ P6 $\oplus$ P7 $\oplus$ P8, R4 = P1 $\oplus$ P3 $\oplus$ P5 $\oplus$ P6, R5 = P3 $\oplus$ P5 $\oplus$ P7, R6 = P5.
5	P0 = A_0B_0, P1 = A_1B_1, P2 = $(A_0 + A_1)(B_0 + B_1)$, P3 = A_2B_2, P4 = $(A_0 + A_2)(B_0 + B_2)$, P5 = A_3B_3, P6 = $(A_0 + A_2 + A_3)(B_0 + B_2 + B_3)$, P7 = A_4B_4, P8 = $(A_2 + A_4)(B_2 + B_4)$, P9 = $(A_1 + A_2 + A_4)(B_1 + B_2 + B_4)$, P10 = $(A_3 + A_4)(B_3 + B_4)$, P11 = $(A_0 + A_1 + A_3 + A_4)(B_0 + B_1 + B_3 + B_4)$, P12 = $(A_0 + A_1 + A_2 + A_3 + A_4)$ $(B_0 + B_1 + B_2 + B_3 + B_4)$.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P4, R3 = P0 $\oplus$ P3 $\oplus$ P5 $\oplus$ P6 $\oplus$ P7 $\oplus$ P8 $\oplus$ P11 $\oplus$ P12, R4 = P0 $\oplus$ P1 $\oplus$ P2 $\oplus$ P5 $\oplus$ P6 $\oplus$ P7 $\oplus$ P9 $\oplus$ P10 $\oplus$ P12, R5 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P4 $\oplus$ P7 $\oplus$ P9 $\oplus$ P11 $\oplus$ P12, R6 = P3 $\oplus$ P5 $\oplus$ P7 $\oplus$ P8, R7 = P5 $\oplus$ P7, R8 = P7.
6	P0 = A_0B_0, P1 = A_1B_1, P2 = $(A_0 + A_1)(B_0 + B_1)$, P3 = $(A_1 + A_2)(B_1 + B_2)$, P4 = $(A_0 + A_1 + A_2)(B_0 + B_1 + B_2)$, P5 = $(A_2 + A_3)(B_2 + B_3)$, P6 = A_4B_4, P7 = $(A_1 + A_4)(B_1 + B_4)$, P8 = $(A_3 + A_4)(B_3 + B_4)$, P9 = $(A_0 + A_1 + A_3 + A_4)(B_0 + B_1 + B_3 + B_4)$, P10 = A_5B_5, P11 = $(A_0 + A_2 + A_5)(B_0 + B_2 + B_5)$, P12 = $(A_0 + A_3 + A_5)(B_0 + B_3 + B_5)$, P13 = $(A_0 + A_2 + A_3 + A_5)(B_0 + B_2 + B_3 + B_5)$, P14 = $(A_4 + A_5)(B_4 + B_5)$, P15 = $(A_1 + A_2 + A_4 + A_5)(B_1 + B_2 + B_4 + B_5)$, P16 = $(A_3 + A_4 + A_5)(B_3 + B_4 + B_5)$.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P2 $\oplus$ P3 $\oplus$ P4, R3 = P1 $\oplus$ P3 $\oplus$ P5 $\oplus$ P6 $\oplus$ P8 $\oplus$ P11 $\oplus$ P13 $\oplus$ P14 $\oplus$ P16, R4 = P1 $\oplus$ P2 $\oplus$ P5 $\oplus$ P7 $\oplus$ P9 $\oplus$ P11 $\oplus$ P13 $\oplus$ P14 $\oplus$ P16, R5 = P0 $\oplus$ P1 $\oplus$ P6 $\oplus$ P7 $\oplus$ P10 $\oplus$ P11 $\oplus$ P12 $\oplus$ P13, R6 = P2 $\oplus$ P4 $\oplus$ P5 $\oplus$ P6 $\oplus$ P7 $\oplus$ P12 $\oplus$ P13 $\oplus$ P14 $\oplus$ P15, R7 = P1 $\oplus$ P2 $\oplus$ P3 $\oplus$ P4 $\oplus$ P5 $\oplus$ P6 $\oplus$ P8 $\oplus$ P12 $\oplus$ P13, R8 = P8 $\oplus$ P14 $\oplus$ P16, R9 = P6 $\oplus$ P10 $\oplus$ P14, R10 = P10.
7	P0 = A_0B_0, P1 = A_1B_1, P2 = $(A_0 + A_1)(B_0 + B_1)$, P3 = A_2B_2, P4 = $(A_0 + A_2)(B_0 + B_2)$, P5 = A_3B_3, P6 = $(A_1 + A_3)(B_1 + B_3)$, P7 = A_4B_4, P8 = $(A_0 + A_4)(B_0 + B_4)$, P9 = A_5B_5, P10 = $(A_3 + A_5)(B_3 + B_5)$, P11 = $(A_1 + A_2 + A_4 + A_5)(B_1 + B_2 + B_4 + B_5)$, P12 = $(A_0 + A_1 + A_3 + A_4 + A_5)(B_0 + B_1 + B_3 + B_4 + B_5)$, P13 = A_6B_6, P14 = $(A_2 + A_6)(B_2 + B_6)$, P15 = $(A_4 + A_6)(B_4 + B_6)$, P16 = $(A_0 + A_1 + A_3 + A_4 + A_6)(B_0 + B_1 + B_3 + B_4 + B_6)$, P17 = $(A_5 + A_6)(B_5 + B_6)$, P18 = $(A_0 + A_1 + A_5 + A_6)(B_0 + B_1 + B_5 + B_6)$, P19 = $(A_0 + A_2 + A_3 + A_5 + A_6)(B_0 + B_2 + B_3 + B_5 + B_6)$, P20 = $(A_1 + A_2 + A_3 + A_5 + A_6)(B_1 + B_2 + B_3 + B_5 + B_6)$, P21 = $(A_0 + A_1 + A_2 + A_3 + A_4 + A_5 + A_6)$ $(B_0 + B_1 + B_2 + B_3 + B_4 + B_5 + B_6)$.	R0 = P0, R1 = P0 $\oplus$ P1 $\oplus$ P2, R2 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P4, R3 = P2 $\oplus$ P3 $\oplus$ P4 $\oplus$ P5 $\oplus$ P6 $\oplus$ P17 $\oplus$ P18 $\oplus$ P19 $\oplus$ P20, R4 = P0 $\oplus$ P1 $\oplus$ P3 $\oplus$ P5 $\oplus$ P6 $\oplus$ P7 $\oplus$ P8, R5 = P0 $\oplus$ P1 $\oplus$ P4 $\oplus$ P5 $\oplus$ P7 $\oplus$ P10 $\oplus$ P11 $\oplus$ P14 $\oplus$ P16, $\oplus$ P17 $\oplus$ P21 R6 = P0 $\oplus$ P3 $\oplus$ P4 $\oplus$ P6 $\oplus$ P7 $\oplus$ P10 $\oplus$ P12 $\oplus$ P13 $\oplus$ P15 $\oplus$ P20 $\oplus$ P21, R7 = P2 $\oplus$ P3 $\oplus$ P5 $\oplus$ P6 $\oplus$ P8 $\oplus$ P9 $\oplus$ P11 $\oplus$ P13 $\oplus$ P15 $\oplus$ P19 $\oplus$ P21, R8 = P3 $\oplus$ P5 $\oplus$ P7 $\oplus$ P9 $\oplus$ P10 $\oplus$ P13 $\oplus$ P14, R9 = P2 $\oplus$ P5 $\oplus$ P7 $\oplus$ P10 $\oplus$ P12 $\oplus$ P15 $\oplus$ P16 $\oplus$ P17 $\oplus$ P18, R10 = P7 $\oplus$ P9 $\oplus$ P13 $\oplus$ P15, R11 = P9 $\oplus$ P13 $\oplus$ P17, R12 = P13.

TABLE II

REALIZATION OF SPACE AND TIME COMPLEXITY METRICS, SUCH AS AREA AND DELAY, IN TERMS OF THE TOTAL NUMBER OF XOR AND AND GATES CONSUMPTION AND CRITICAL PATH DELAY OF THE MULTIPLIERS

M, n	No. of sub multipliers	Space Complexity	Time complexity
2	3	$S_{XOR} = 3n - 2 + 3(MTK(\frac{n}{2}))$, (8) $S_{AND} = 3^{Rec} \times 5$. (10)	$T_{Total} = ((3 \times Rec) + 3)T_G$. (9)
3	6	$S_{XOR} = (\frac{22n}{3}) - 10 + 6(MTK(\frac{n}{3}))$, (11) $S_{AND} = 6^{Rec} \times 5$. (13)	$T_{Total} = ((6 \times Rec) + 3)T_G$. (12)
4	9	$S_{XOR} = 29n - 24 + 9(MTK(\frac{n}{4}))$, (14) $S_{AND} = 9^{Rec} \times 5$. (16)	$T_{Total} = ((12 \times Rec) + 3)T_G$. (15)
5	13	$S_{XOR} = (\frac{102n}{5}) - 40 + 13(MTK(\frac{n}{5}))$, (17) $S_{AND} = 13^{Rec} \times 5$. (19)	$T_{Total} = ((13 \times Rec) + 3)T_G$. (18)
6	17	$S_{XOR} = 25n - 57 + 17(MTK(\frac{n}{6}))$, (20) $S_{AND} = 17^{Rec} \times 5$. (22)	$T_{Total} = ((12 \times Rec) + 3)T_G$. (21)
7	22	$S_{XOR} = (\frac{220n}{7}) - 80 + 22(MTK(\frac{n}{7}))$, (23) $S_{AND} = 22^{Rec} \times 5$. (25)	$T_{Total} = ((17 \times Rec) + 3)T_G$. (24)

partial products are calculated using submultipliers followed by reconstruction step. The coefficients of the multiplication result polynomial are presented as $R_0, R_1, \ldots, R_C$, where

$$C = 2M - 2. \tag{26}$$

Using the equations in Table I, the theoretical boundaries for XOR and AND gates space and time complexities were determined and presented in Table II where the following holds.

1) S_{XOR}*:* Number of XOR gates required/additive cost.
2) S_{AND}*:* Number of AND gates required/multiplicative cost.
3) T_{Total}*:* Combinational delay required to multiply the given operands.
4) T_G*:* Assume that both the combinational delay of AND and XOR gates are the same.
5) Rec*:* Number of recurrent stage required.

The total space complexity of the multiplier could be calculated using the following equation:

$$S_{\text{Total}} = S_{\text{XOR}} + S_{\text{AND}}. \tag{27}$$

In Table II, Rec denotes the number of recurrence stages required. The main advantage of employing M-term Karatsuba-like functions is that it reduces the number of recurrence stages compared to KOM.

C. Discussion of Theoretical Complexities

The time and space complexities for M-term Karatsuba-like multipliers depend on the number of stages and submultipliers.

It is obvious that, for each operand size of "n," the number of recurrence stages required for larger values of "M" is smaller than the number of recurrent stages necessary for a two-term Karatsuba-like multiplier.

Since each stage adds an extra delay to the total delay of the multiplier, it is expected that the delay of the two-term Karatsuba-like would be larger than multipliers with larger M.

On the other hand, the size of the multiplier is smaller in two-term as it only has three submultipliers. For example, in the seven-term multiplier, 22 submultipliers have been employed, and as a result, it has a larger area than that of the two-term. The total number of gates (XOR and AND) and their corresponding delay are considered for theoretical evaluation of complexity. It is evident that the performance of each M-term Karatsuba-like varies depends on the size of the operands. However, the total number of gate utilization significantly increases when the number of recurrence iterations drops. Every term M-term Karatsuba-like multipliers were accessed in Section III for different operand sizes, and the performance metrics, such as space and time complexities, were examined with a detailed comparison of theoretical data and FPGA implementation results.

III. PROPOSED METHOD

This work takes advantage of the low time complexity in SBM and low-space complexity in M-term Karatsuba by combining these two methods. Furthermore, the area–delay tradeoff was investigated.

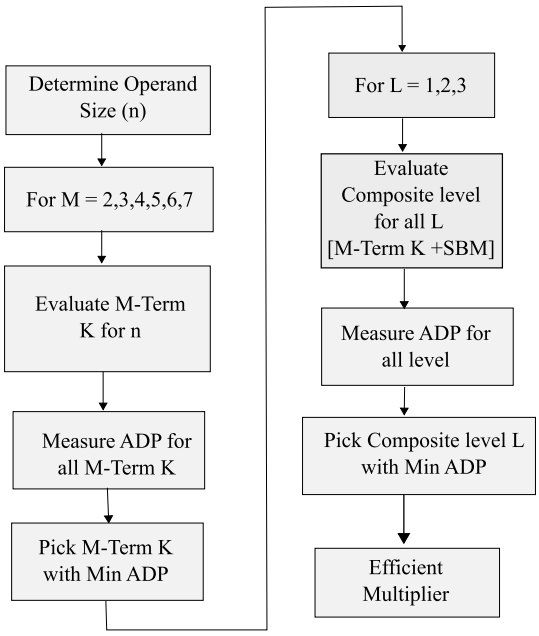

Fig. 1. Proposed methodology to achieve efficient multiplier using M-term Karatsuba-like and SBM. ADP: area–delay product.

Compared to the school book, M-term Karatsuba has a disadvantage of higher time complexity due to the larger number of the recursive product. However, it is considerably more efficient because of the lower area–delay product (ADP).

A methodology to achieve an efficient composite finite field multiplier is given in Fig. 1. This method utilizes the M-term Karatsuba-like multiplier at the upper bound and SBM at the lower bound of recurrent stages. It reduces the higher time complexity of the M-term Karatsuba-like multiplier and optimizes higher combinational delay.

The implementation of this methodology was handled in two phases.

1) *Phase A:* The M-term Karatsuba-like multiplier for operand size of "n" was implemented for $M = 2$, 3, 4, 5, 6, and 7. Based on the ADP, an efficient M-term multiplier was selected.
2) *Phase B:* A composite multiplier for the operand size of "n" was constructed for different levels of 1, 2, and 3. The composite M-term multiplier with a minimum ADP was adopted as the most efficient multiplier.

The same procedure was used to find the most efficient multiplier for various operand sizes from $n = 8$ to 750, including some of the operand sizes recommended by the National Institute Standards and Technology (NIST), such as GF(2^{233}), GF(2^{283}), and GF(2^{409}).

A. Efficient M-term Karatsuba-Like Functions

First, hardware description language (HDL) codes were developed for each multiplier according to the recursive products and reconstruction functions represented in Table I. A set of python frameworks for each split function was created where the frameworks automate the generation of the VHDL codes for operand size n for all the M-term Karatsuba-like functions. Thereafter, HDL codes were synthesized on Xilinx Artix 7 FPGA (XC7A35TIC) device. To measure the delay, the input/output (i/o) buffer was neglected during the experimental study. This FPGA family contains six input LUTs as building blocks.

The data reported in this section are only for implementation on Xilinx Artix 7 FPGA. It is worth mentioning that these results could change according to FPGA device or synthesis tools since each tool has a set of setting, which can optimize results for the area, speed, power, and other criteria. Besides, different vendors have various algorithms for optimization with different boundary conditions.

The total number of LUTs, combinational delay, and ADP for hardware implementation of M-term Karatsuba-like multiplier for different operand sizes are presented in Figs. 3–5(a)–(f). It needs to be acknowledged that the implementation results presented in this section do not include the delay and area for modular reduction operation. The reduction module adds a constant value of delay and later would be added to construct a finite field multiplier.

The two-term function has been employed at all the last recurrent stages to reduce the zero paddings on some split functions. For example, if the operand size is 2, and if it needs to be reduced by a seven-term Karatsuba-like, there is a need to pad five zeros with the given operand, and hence, it increases the computing complexities.

1) Area Complexity Comparison: In Fig. 2, the trend variations of area complexity, such as LUTs and theoretical gates to implement the all M-term Karatsuba-like multipliers, are illustrated. LUT complexities are recorded from the circuit implemented on the Xilinx Artix 7 FPGA, whereas the gate complexities were evaluated using Table II.

From Fig. 2, it is visible that the number of gates and LUT utilizations increase when the operand size increases. Our implementation results confirm that the area complexity mainly depends on the number of recursive products and recurrent stages (Rec) used to break down the given operands. It is worth noting that the area required to compute the two-term Karatsuba-like multiplier is smaller than the seven-term Karatsuba-like for the given operand size. This is because the two-term uses only three recursive products, whereas the seven-term uses 22 recursive products.

The area complexity of other M-term Karatsuba-like lies between two-term and seven-term, and it mainly depends on the number of recurrent stages for the operand size. From Table III, it is evident that there is a hike in LUT utilization whenever there is an increase in the number of recurrent stages (Rec). For example, for $M = 2$, the difference in LUT consumption is not huge between $n = 232$ and 282 since it requires eight recurrent stages to compute the product. However, when the operand size increased to 409, there is a 185% increase in LUT since it requires nine recurrent steps. As the operand size grows, the same trend was experienced up to the operand size $n = 750$.

2) Time Complexity Comparison: In Fig. 3, the trend for time complexity, such as combinational delay and the theoretical gate path for all M-term Karatsuba-like multipliers, is plotted. Delay complexities are also recorded on the Xilinx

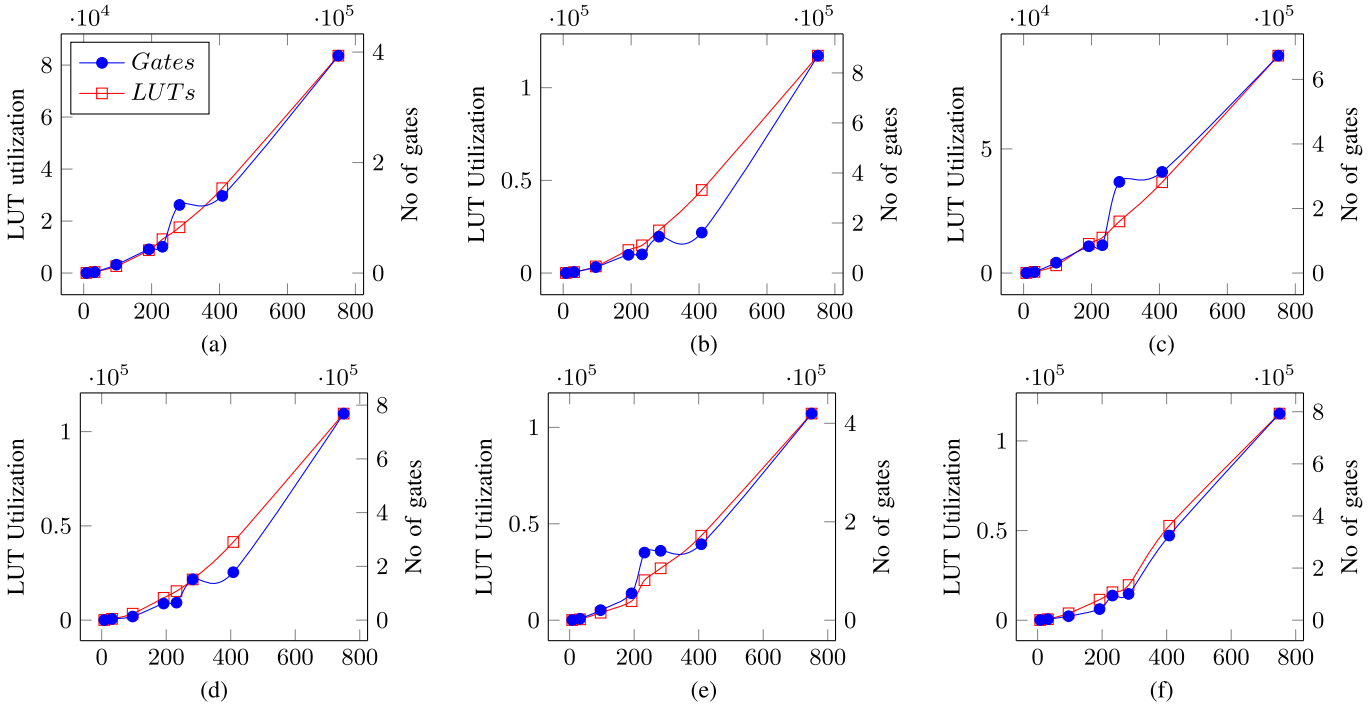

Fig. 2. Comparing area complexities of theoretical study and results for practical implementation of M-term Karatsuba-like multipliers on FPGA in terms of the number of gates and LUTs for each operand size of (n). (a) Two-term multiplier. (b) Three-term multiplier. (c) Four-term multiplier. (d) Five-term multiplier. (e) Six-term multiplier. (f) Seven-term multiplier.

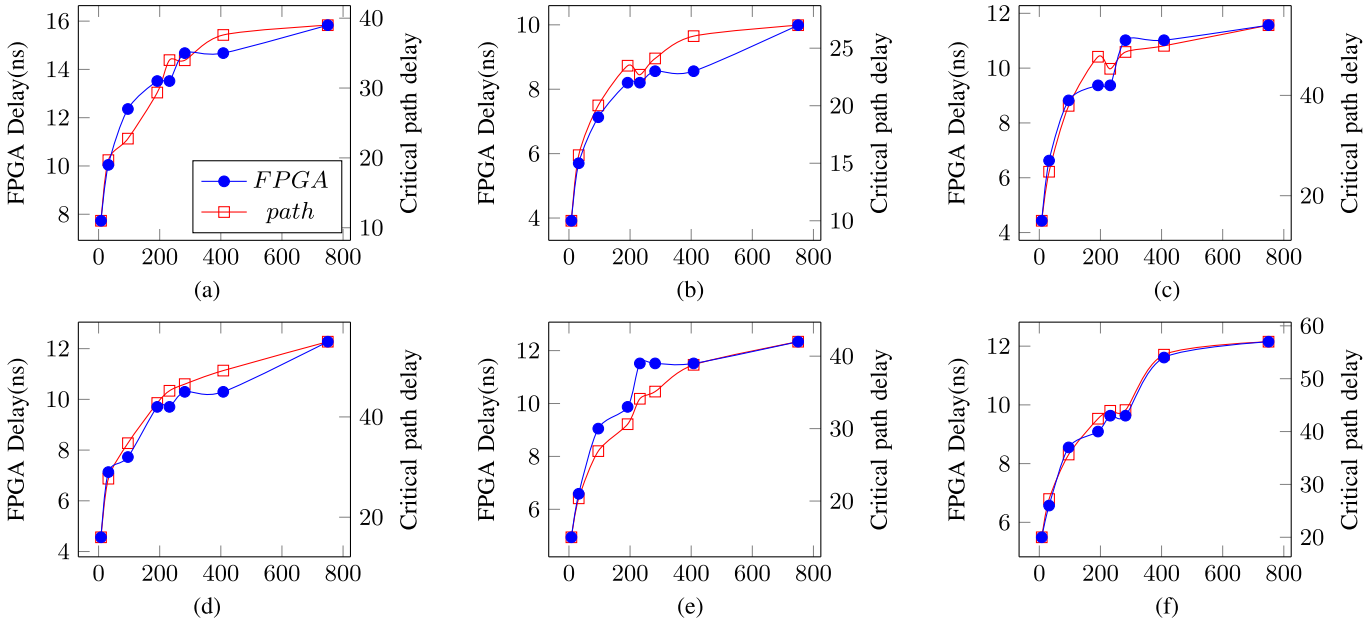

Fig. 3. Comparing time complexities obtained from theoretical study and results for practical implementation of M-term Karatsuba-like multipliers on FPGA in terms of the number of gates and LUTs for each operand size of (n). (a) Two-term multiplier. (b) Three-term multiplier. (c) Four-term multiplier. (d) Five-term multiplier. (e) Six-term multiplier. (f) Seven-term multiplier.

Artix 7 FPGA, whereas the gate path complexities were analyzed using Table II. The graph shows that the FPGA data recorded from the implementation match the theoretical evaluation.

As previously mentioned in the area complexity analysis, the number of recurrent stages (Rec) still plays a significant role in determining the curvature of the combinational delay. This is confirmed in Table IV.

From Table IV, it is obvious that the number of the recurrent stages (Rec) and the critical path delay of each stage are considered important factors that determine the speed of the multiplication. The two-term has a higher number of Rec,

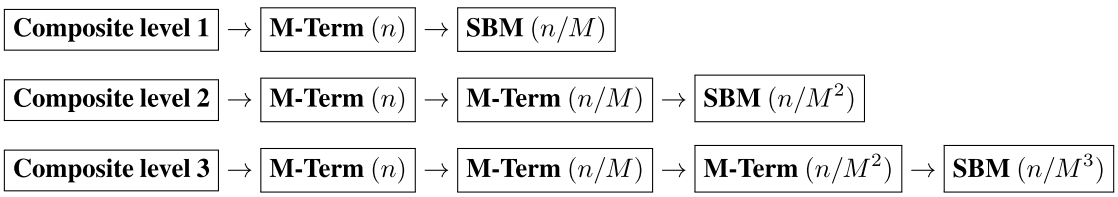

Fig. 4. Proposed composite levels using M-term Karatsuba-like functions and SBM for the operand size n. Three levels have been introduced to achieve an optimized multiplier. Level 1: M-term followed by SBM. Level 2: two iterative M-term followed by SBM. Level 3: three iterative M-term followed by SBM. M-term Karatsuba utilized.

TABLE III

NUMBER OF LUTS USED AND NUMBER OF RECURRENT STAGES (REC) REQUIRED TO COMPUTE THE MULTIPLICATION FOR THE GIVEN OPERAND SIZE $n = 232$, 282, AND 409 IN XILINX ARTIX 7 FPGA

	n = 232		n = 282		n = 409	
M	LUT	Rec	LUT	Rec	LUT	Rec
2	13010	8	17619	8	32631	9
3	14951	5	22995	5	44798	6
4	14286	4	20801	4	36590	5
5	15378	4	21574	4	41463	5
6	20758	4	269301	4	43817	4
7	15614	3	19620	3	52680	4

TABLE IV

COMBINATIONAL DELAY AND NUMBER OF RECURRENT STAGES (REC) REQUIRED TO COMPUTE THE MULTIPLICATION FOR THE GIVEN OPERAND SIZE $n = 232$, 282, AND 409 IN XILINX ARTIX 7 FPGA

	n = 232		n = 282		n = 409	
M	Delay	Rec	Delay	Rec	Delay	Rec
2	14.4	8	14.4	8	15.4	9
3	8.5	5	8.9	5	9.6	6
4	9.9	4	10.5	4	10.8	5
5	10.3	4	10.6	4	11.1	5
6	10.2	4	10.5	4	11.5	4
7	9.7	3	9.8	3	11.70	4

whereas each Rec stage has a lower critical path. Though seven-term has lower Rec stages for $N = 232$, 282, and 409, the FPGA delays are not lesser than three-term because of the higher critical path delay of each Rec stage. The two-term function has less area utilization; on the other hand; it requires more delay because of the extensive recurrent stage (Rec). For $n = 232$ multiplication, two-term requires 14.4 ns, and seven-term requires only 9.7 ns, hence seven-term, which is 67% faster for given operand size. To understand the performance behavior of the M-term Karatsuba-like for higher operand sizes, area and delay complexity analyses were extended up to $n = 2500$. Then, LUTs and combinational delays were recorded, and the ADP was computed to pick the efficient M-term Karatsuba-like functions for each operand size.

B. Optimization Using Proposed Method

After implementing the M-term Karatsuba-like for $n = 232$ to 2500, the efficient M-term multiplier is chosen for each operand size (n). The ADPs, along with the efficient M-term Karatsuba-like corresponding to the operand sizes, were tabulated in Table V. The main advantage of Karatsuba over SBM is having a lower area complexity. The tradeoff, however, is an increase in time complexity. The delay of M-term Karatsuba-like multipliers required is competitive to SBM, yet it does not suffer from the very high space complexity of the SBM. It is evident that the M-term Karatsuba-like multipliers have better performance in the higher bound of the recurrent stage (Rec), and SBM is more efficient in lower bounds. To make the multiplier more efficient, the composite model was presented by investigating the tradeoff between the area and time complexities.

TABLE V

EFFICIENT M-TERM KARATSUBA-LIKE FUNCTIONS AND THEIR LUTS, COMBINATIONAL DELAY, AND ADP ON XILINX SPARTAN 7 FOR VARIOUS OPERAND SIZES

n	Efficient M-Term	LUT	Delay	ADP
232	3-Term	14951	6.55	97959
282	7-Term	19620	7.71	151309
409	4-Term	36590	8.59	314308
750	4-Term	87549	9.12	798797
1000	4-Term	144010	9.55	1376160
1350	3-Term	279195	8.40	2351957
1800	5-Term	390998	10.58	4136759
2200	4-Term	498712	10.95	5461894
2500	5-Term	632549	11.01	6965630

Phase B introduces the composite method with different levels by combining SBM with M-term Karatsuba-like multipliers. Combined designs were developed by using the M-term function on top levels and SBM on bottom levels.

Fig. 4 demonstrates the different levels of composite level explorations for an operand size n. In level 1, the M-term Karatsuba function is implemented in the first recurrent stage for operand size n, and the single-step SBM is implemented on the next recurrent step (n/M). In level 2, two M-term functions have been employed on the top-two recurrent stages continued by SBM with a size of (n/M^2). In level 3, SBM is implemented in the fourth recurrent stage with a size of (n/M^3).

All of these levels were tested on Xilinx Spartan 7, and performance metrics were recorded. Table VI summarizes the results for all three composite levels based on the ADP. As the data in Table VI show, after finding the efficient M-term Karatsuba-like, our proposed method was able to find the more

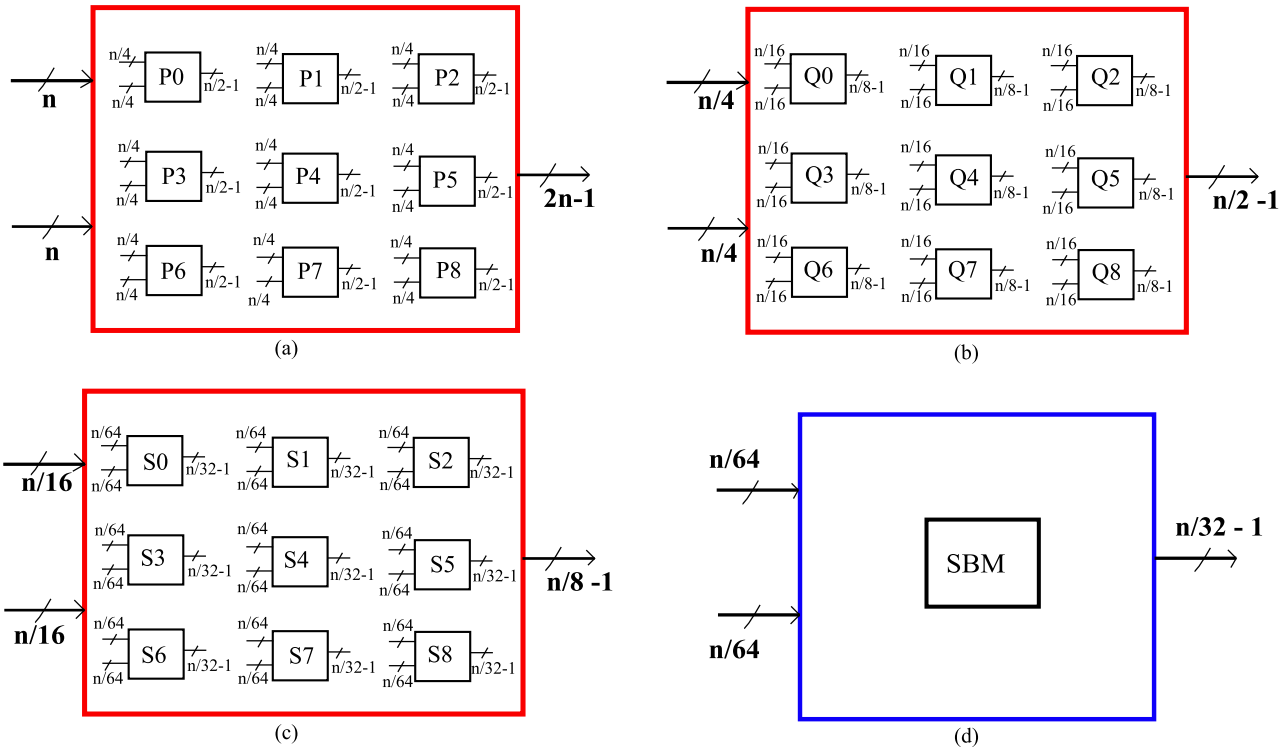

Fig. 5. Top level to inner level blocks of hardware implementation of four-term composite level 3 multiplier. A and B are the two operand used with corresponding operand size for each level. P0–P8: submultipliers of top level with operand size of $n/4$. Q0–Q8: submultipliers of second most level with operand size of $n/16$. S0–S8: submultipliers of third most level with operand size of $n/64$. (a) Top-level four-term. (b) Expansion of submultipliers [Px]. (c) Expansion of submultipliers [Qx]. (d) Expansion of submultipliers [Sx].

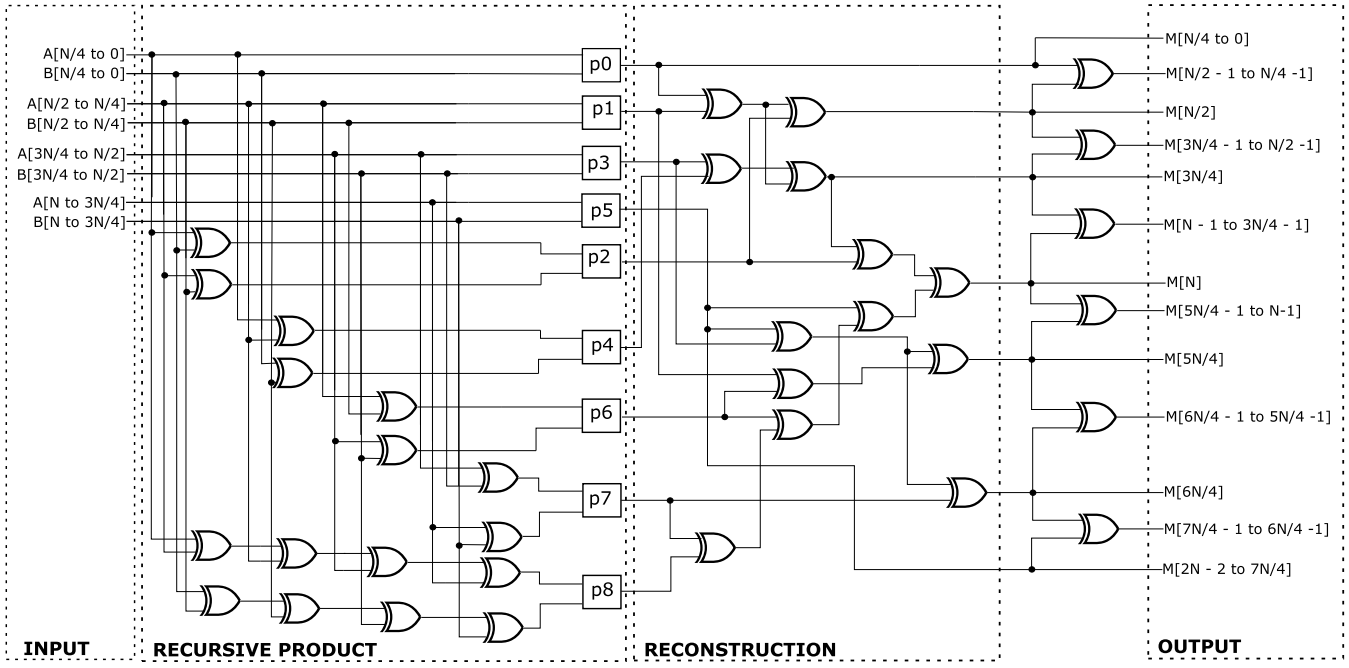

Fig. 6. Gate-level design for hardware implementation of n-bit four-term Karatsuba-like multiplication demonstrated in Fig. 5. The data flow graph was categorized into input, recursive product, reconstruction, and output. In this design, A and B are the inputs with operand size n, and M is the output from the four-term Karatsuba-like multiplier with bit size $2n - 1$.

efficient multiplier, which outperforms the current findings. It was confirmed with the experimental evaluation, and the highlighted data show the efficient composite levels for each operand size.

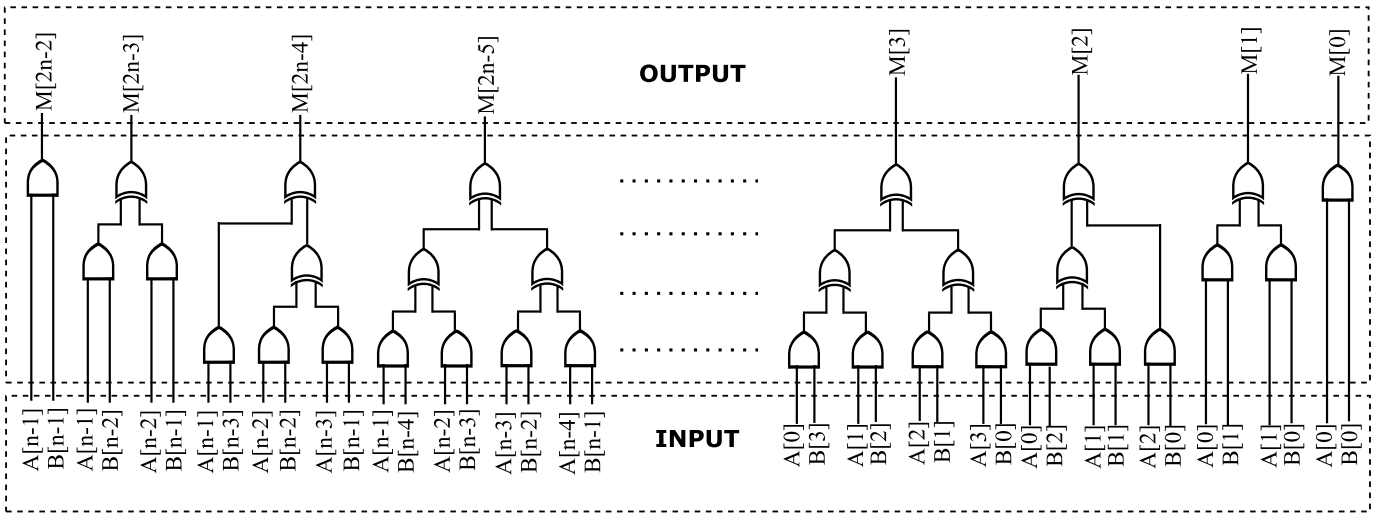

Fig. 7. Gate-level design for hardware implementation of n-bit SBM demonstrated in Fig. 5. The data flow graph was categorized into input, logic block, and output. In this design, A and B are the inputs with operand size n, and M is the output from the school-book multiplier with bit size $2n-1$.

TABLE VI
NUMBER OF LUTS AND COMBINATIONAL DELAY REQUIRED FOR EACH COMPOSITE LEVEL ON XILINX SPARTAN 7 FPGA (XC7S100)

Operand	**Efficient M-Term**		**Optimization Level 1**			**Optimization Level 2**			**Optimization Level 3**		
	M-Term	ADP	LUT	Delay	ADP	LUT	Delay	ADP	LUT	Delay	ADP
232	**3**	97959	16915	5.35	90546	**15617**	**5.68**	**88720**	*	*	*
282	**7**	151309	**18042**	**6.06**	**109353**	19086	7.68	146542	*	*	*
409	**4**	314308	**40056**	**6.17**	**247266**	36899	7.44	274344	30128	8.48	255516
750	**4**	798797	125375	6.18	775821	114817	7.58	870657	**85777**	**8.55**	**733479**
1000	**4**	1376160	218689	6.78	1484242	153030	7.78	1189808	**129430**	**8.75**	**1132254**
1350	**3**	2351957	464386	6.33	2943278	329780	6.73	2220739	**286200**	**7.38**	**2113015**
1800	**5**	4136759	646268	6.95	4496733	**402017**	**8.27**	**3324279**	355652	9.71	3453381
2200	**4**	5461894	1077197	7.10	7648099	**637527**	**8.04**	**5126992**	565676	9.24	5226281
2500	**5**	6965630	1292241	7.08	9150359	739746	8.56	6334445	**618850**	**9.70**	**6003464**

TABLE VII
COMPARISON OF THE PROPOSED WORK WITH RELEVANT STUDIES IN TERMS OF FPGA RESOURCE UTILIZATION AND DELAY OF THE MULTIPLIER

Operand	**Similar works**	**Algorithm**	**LUTs**	**Delay**	**ADP**	**Device/Sythesis Tool**	**Reduction**
8	Samanta et al [30]	Modified Karatsuba	62	13.95	1367	Spartan-3E XC3S100E	No
	Our work	Composite	26	4.91	128	Xilinx ISE	
8	Mathe et al [39]	Modified Systolic	98	10.55	1034	Virtex 7 XC7V2000TF	Yes
	Our work	Composite	31	2.52	78	Xilinx Vivado	
233	Rashidi et al [40]	Pipelined Bit-parallel	36812	7.19	264678	Virtex 5 XC5VLX110	Yes
	Our work	Composite	20802	8.45	175714	Xilinx ISE	
233	Xie et al [29]	Digit serial	1420	49.50	70290	Stratix II EP2S180F	Yes
	Our work	Composite	17729	11.01	195196	Intel Quartus Prime	
233	Fan et al [28]	Overlap-free Karatsuba	14255	6.34	90316	Spartan 7 XC7S100	Yes
	Our work	Composite	15617	5.68	88720	Xilinx Vivado	
283	Zhou et al [41]	Modified Karatsuba	15525	8.18	126917	Virtex 5 XC5VLX110	Yes
	Our work	Composite	19802	6.50	128713	Xilinx ISE	
408	Dhanaselvam et al [42]	Recursive Karatsuba	32679	9.60	313555	Spartan 7 XC7S100	No
	Our work	Composite	40056	6.17	247265	Xilinx Vivado	
750	Gaudal et al [38]	N-way Split Karatsuba	87549	9.12	798797	Spartan 7 XC7S100	No
	Our work	Composite	85777	8.55	733479	Xilinx Vivado	

From this analysis, delays of composite levels are smaller compared to the efficient M-term method because of the SBM used in the lower stage. At composite level I, SBM is used at the next stage of the top-level M-term Karatsuba-like, and it is apparent that it has the lowest delay of all. At level II, our model has slightly more delay than level I because of the

TABLE VIII

LUT INCREMENT OF M-TERM FUNCTIONS AND OUR PROPOSED METHOD COMPARED TO GENERAL KOM. "+" SIGN REPRESENTS THE INCREASE AND "−" REPRESENTS THE DECREASE IN LUT USAGE IN PERCENTAGE

LUT Increase (%)		
Operand	M-Term Multiplier	Proposed work
232	+14.7	+19.9
282	+11	+2.1
409	+11.9	+22.6
750	+4.7	+2.6
1000	+8.5	**-2.5**
1350	+22.9	+25.6
1800	+8.7	+11.7
2200	-3.8	+23
2500	+4.3	+2

TABLE IX

DELAY REDUCTION OF M-TERM KARATSUBA-LIKE AND OUR PROPOSED METHOD COMPARED TO GENERAL KOM. "−" REPRESENTS THE DECREASE IN DELAY IN PERCENTAGE

Delay Reduction (%)		
Operand	M-Term Multiplier	Proposed work
232	-26.7	**-36.4**
282	-8.8	**-28.3**
409	-10.5	**-35.7**
750	-4.7	**-10.7**
1000	-10.5	**-18.1**
1350	-21	**-30.6**
1800	-11.8	**-31.1**
2200	-4.1	**-29.6**
2500	-6	**-17.2**

two iterative steps of MTK. Area complexity trends are vice versa. When the delays decrease, area consumption becomes slightly larger since we use larger SBM in level I and smaller SBM in different levels. However, here, the proposed method still outperforms the efficient split method in terms of ADP.

Fig. 5 demonstrates the hardware description blocks from the top level to the inner logic level. For the illustration of the proposed method, four-term composite level 3 was used with the operand size n. The top-level four-term Karatsuba-like has two input operand A and B of bit length n and has an output of M with $[2n-2]$ bit length. The next two levels of four-term Karatsuba-like have $n/4$ and $n/16$ correspondingly. The SBM has been used at a lower level with the operand size $n/64$. In Fig. 6, the gate-level design of four-term Karatsuba-like is depicted for operand A and B with size n. The design uses nine recursive products and reconstruction, as mentioned in Table I, to generate the product M with bit length $[2n-1]$.

Fig. 7 depicts the data-flow graph (DFG) of gate-level design for hardware implementation of n-bit SBM. The input operands are A and B with the size of n and the output of M with the operand size of $2n-1$.

For the smaller operand sizes, such as $n = 232$ to 409, composite level I is more efficient than other levels. In contrast, levels II and III give better performance for operand sizes $n = 750$ to 2500.

IV. RESULT AND DISCUSSION

The device utilization and speeds mentioned in Section III so far do not include I/O buffers. However, the proposed method was further implemented with I/O buffers enabled when it was compared with similar works with I/O buffers enabled. The reduction modules were treated in the same way. The irreducible NIST polynomials, such as trinomial: $x^{233}+x^{74}+1$, and pentanomials, such as $x^8+x^4+x^3+1$ and $x^{283}+x^{12}+x^7+x^5$, were used for modular reduction to compare with similar works.

The result of the proposed work was compared with similar work using the same FPGA device used in those works including Xilinx and Intel FPGAs, and presented in Table VII.

It should be noted that each FPGA chip is fabricated with different technologies, and the circuit might give considerable variations in terms of LUTs and delay. It is worth noting that many similar works had been studied in a specific FPGA, and then, the same FPGA was used for implementing our proposed method to preserve the accuracy.

Mathe and Boppana [38] had proposed a low power and low hardware modified systolic multiplier for NIST irreducible pentanomial: $x^8+x^4+x^3+1$. This design was implemented in Xilinx Virtex 7 FPGA, and it consumes 98 LUTs, whereas our proposed method requires 98 LUTs. Our design has a considerably low combinational delay and resulting in low ADP. Samanta *et al.* [30] had proposed a modified Karatsuba multiplier (MKM) on the FPGA platform. The method introduced an alternate splitting method and claimed that the MKM performs better than standard KOM in speed and area. This work utilizes 1367 ADP in Xilinx Spartan, whereas our proposed method uses only 128 ADP to compute 8-bit multiplication.

Rashidi [39] proposed a pipelined version of bit-parallel multipliers. It has more significant area consumption and results in high ADP requirements. In Xie *et al.*'s work [29], the two operands are multiplied in digit level to decrease the area requirement for the multiplier for irreducible trinomials: $x^{233}+x^{74}+1$. However, it requires extensive delay to compute all the operations through digit serial Karatsuba. However, the method has fewer hardware requirements, our proposed way is more efficient in terms of speed, and it is verified on Xilinx Virtex 5. The comparison was presented to explain the delay reduction trend from 49.5 to 11.1 ns.

Fan *et al.* [28] presented overlap-free strategy on the Karatsuba multiplier. The performance of this multiplier evaluated in Xilinx Spartan 7 is nearly similar, but the ADP utilized for the 233-bit operand is higher than our proposed method.

For operand size 283, our method has lower delay requirements compared to Zhou *et al.* [40] Karatsuba modification for irreducible pentanomial: $x^{283}+x^{12}+x^7+x^5$. It was tested in Xilinx Virtex 5 with modular reduction without including I/O buffer, as given in their research. Similarly, for 408-bit operands, recursive Karatsuba of Dhanaselvam *et al.* [41] was

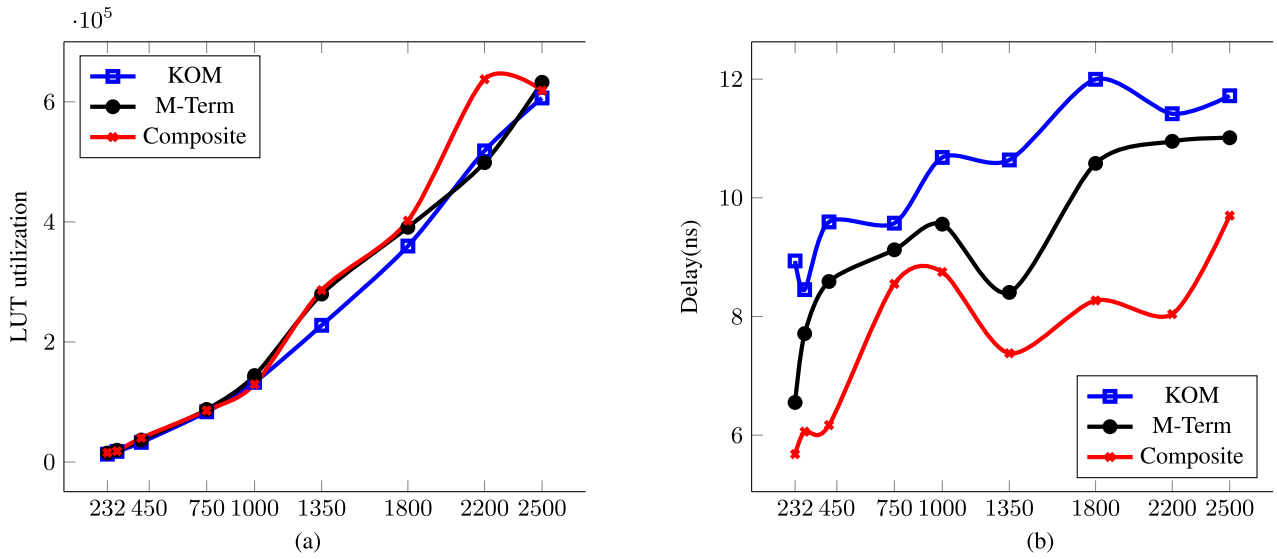

Fig. 8. Graph explains the trends of practically recorded (a) LUTs and (b) combinational delay of standard KOM, M-term Karatsuba-like, and the proposed method on Xilinx Spartan 7 FPGA.

TABLE X

SPEED IMPROVEMENTS OF PROPOSED MULTIPLIER ACHIEVED FOR THE NIST OPERAND SIZES n, SUCH AS 96, 192, 232, 282, AND 409. THE THEORETICAL TIME COMPLEXITIES FOR THOSE OPERAND SIZES WERE COMPARED WITH STANDARD KARATSUBA AND THREE-WAY KARATSUBA MULTIPLIERS. **PROPOSED MULTIPLIER* = EFFICIENT MULTIPLIER DESIGN ACHIEVED FROM THE PROPOSED STRUCTURE DISCUSSED IN FIG. 1**

n	**96**	**192**	**232**	**282**	**409**
2-way KM [28]	19 T_x + 1 T_a	21 T_x + 1 T_a	22 T_x + 1 T_a	23 T_x + 1 T_a	25 T_x + 1 T_a
3-way KM [43]	16 T_x + 1 T_a	18 T_x + 1 T_a	19 T_x + 1 T_a	20 T_x + 1 T_a	21 T_x + 1 T_a
Proposed Multiplier*	11 T_x + 1 T_a	15 T_x + 1 T_a	16 T_x + 1 T_a	20 T_x + 1 T_a	19 T_x + 1 T_a

compared, and our method has higher performance than that work.

The experimental evaluation tabulated in Tables VIII and IX show the average values of improvements by M-term Karatsuba-like and proposed work compared to standard KOM. As shown in Table VIII, the LUT utilizations of M-term Karatsuba-like and the proposed method are slightly larger than standard KOM. The average LUT increment of the M-term function over KOM is 9.2% on average since it has a 13% average drop in delay and makes the M-term multiplier more efficient than the standard KOM, while the proposed method consumes 11% of additional area, reduces 26% of delay compared to standard KOM, and collectively gives a 15% reduction in ADPs compared to the standard KOM. From Table IX, it is evident that the proposed method is more efficient than the standalone M-term Karatsuba-like method in terms of combinational delay.

Fig. 8 demonstrated the LUT utilization and delay overheads between standard KOM, M-term Karatsuba-like, and our proposed method. For each operand size, the trend varies for both the metrics; however, the delay of the composite method is always low and makes the proposed method more efficient in terms of ADP.

Table X demonstrates the speed improvements of the proposed multiplier over existing methods. This table tabulates the theoretical delay components utilized for designing the n bit proposed multiplier over two-way Karatsuba and three-way Karatsuba multiplication algorithms. Due to the proposed methodology, our design outperforms the existing method with a considerably low combinational delay. This delay reduction was achieved by eliminating the excess Rec stages by employing SBM on lower levels. This theoretical demonstration shows that the proposed multiplier is nearly 25% faster on average, validated from the experimental evaluation on different FPGA devices.

V. CONCLUSION

In this article, first M-term Karatsuba-like binary multipliers were analyzed in terms of space and time complexities for different values of M and various operand sizes (n). Performance parameter's trends were pictured for the Xilinx Artix FPGA device. Later, a novel composite method is introduced to take advantage of the low-space complexity of M-term Karatsuba-like and low time complexity SBM. The proposed method was extensively tested on different FPGAs to attain the improvement graph over other similar works. In FPGA devices, implementation results show that the composite method requires 11% additional resources and drops the delay complexity 26% lower, and it is 15% more efficient in ADP than standard KOM. This work achieved the suitable tradeoff between space and time complexities, which minimizes the ADP requirement of the multiplier.

REFERENCES

[1] R. Abu-Salma, M. A. Sasse, J. Bonneau, A. Danilova, A. Naiakshina, and M. Smith, "Obstacles to the adoption of secure communication tools," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2017, pp. 137–153.

[2] B. Vembu, A. Navale, and S. Sadhasivan, "Creating secure communication channels between processing elements," U.S. Patent 9 589 159, Mar. 7, 2017.

[3] J. Yoo and J. H. Yi, "Code-based authentication scheme for lightweight integrity checking of smart vehicles," *IEEE Access*, vol. 6, pp. 46731–46741, 2018.

[4] P. Aparna and P. V. V. Kishore, "Biometric-based efficient medical image watermarking in E-healthcare application," *IET Image Process.*, vol. 13, no. 3, pp. 421–428, 2019.

[5] T. D. Premila Bai, K. M. Raj, and S. A. Rabara, "Elliptic curve cryptography based security framework for Internet of Things (IoT) enabled smart card," in *Proc. World Congr. Comput. Commun. Technol. (WCCCT)*, Feb. 2017, pp. 43–46.

[6] Z. U. A. Khan and M. Benaissa, "High-speed and low-latency ECC processor implementation over GF(2^m) on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 25, no. 1, pp. 165–176, Jan. 2017.

[7] G. Chen, G. Bai, and H. Chen, "A high-performance elliptic curve cryptographic processor for general curves over GF(p) based on a systolic arithmetic unit," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 54, no. 5, pp. 412–416, May 2007.

[8] H. Marzouqi, M. Al-Qutayri, K. Salah, D. Schinianakis, and T. Stouraitis, "A high-speed FPGA implementation of an RSD-based ECC processor," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 24, no. 1, pp. 151–164, Jan. 2016.

[9] K. C. C. Loi and S. B. Ko, "Scalable elliptic curve cryptosystem FPGA processor for NIST prime curves," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 11, pp. 2753–2756, Jan. 2015.

[10] N. Y. Goshwe, "Data encryption and decryption using RSA algorithm in a network environment," *Int. J. Comput. Sci. Netw. Secur. (IJCSNS)*, vol. 13, no. 7, p. 9, 2013.

[11] R. J. McEliece, *Finite Fields for Computer Scientists and Engineers*, vol. 23. Boston, MA, USA: Springer, 2012.

[12] N. Homma, K. Saito, and T. Aoki, "Toward formal design of practical cryptographic hardware based on Galois field arithmetic," *IEEE Trans. Comput.*, vol. 63, no. 10, pp. 2604–2613, Oct. 2014.

[13] A. D. Piccoli, A. Visconti, and O. G. Rizzo, "Polynomial multiplication over binary finite fields: New upper bounds," *J. Cryptograph. Eng.*, vol. 10, pp. 197–210, Apr. 2019.

[14] F. Mallouli, A. Hellal, N. Sharief Saeed, and F. Abdulraheem Alzahrani, "A survey on cryptography: Comparative study between RSA vs ECC algorithms, and RSA vs el-gamal algorithms," in *Proc. 6th IEEE Int. Conf. Cyber Secur. Cloud Comput. (CSCloud)/5th IEEE Int. Conf. Edge Comput. Scalable Cloud (EdgeCom)*, Jun. 2019, pp. 173–176.

[15] B. Sunar, "A generalized method for constructing subquadratic complexity GF (2^k) multipliers," *IEEE Trans. Comput.*, vol. 53, no. 9, pp. 1097–1105, Sep. 2004.

[16] K.-W. Kim and J.-C. Jeon, "Polynomial basis multiplier using cellular systolic architecture," *IETE J. Res.*, vol. 60, no. 2, pp. 194–199, 2014.

[17] C.-Y. Lee, C.-C. Chen, Y.-H. Chen, and E.-H. Lu, "Low-complexity bit-parallel systolic multipliers over GF(2^m)," in *Proc. IEEE Int. Conf. Syst., Man Cybern.*, vol. 2, Oct. 2006, pp. 1160–1165.

[18] W. Tan, A. Au, B. Aase, S. Aao, and Y. Lao, "An efficient polynomial multiplier architecture for the bootstrapping algorithm in a fully homomorphic encryption scheme," in *Proc. IEEE Int. Workshop Signal Process. Syst. (SiPS)*, Oct. 2019, pp. 85–90.

[19] W. Liu, S. Fan, A. Khalid, C. Rafferty, and M. O'Neill, "Optimized schoolbook polynomial multiplication for compact lattice-based cryptography on FPGA," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 10, pp. 2459–2463, Oct. 2019.

[20] C. Rafferty, M. O'Neill, and N. Hanley, "Evaluation of large integer multiplication methods on hardware," *IEEE Trans. Comput.*, vol. 66, no. 8, pp. 1369–1382, Aug. 2017.

[21] Z. Gu and S. Li, "A division-free Toom-Cook multiplication-based Montgomery modular multiplication," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 66, no. 8, pp. 1401–1405, Aug. 2019.

[22] J. Ding and S. Li, "A low-latency and low-cost Montgomery modular multiplier based on NLP multiplication," *IEEE Trans. Circuits Syst. II, Exp. Briefs*, vol. 67, no. 7, pp. 1319–1323, Jul. 2020.

[23] D. Zoni, A. Galimberti, and W. Fornaciari, "Flexible and scalable FPGA-oriented design of multipliers for large binary polynomials," *IEEE Access*, vol. 8, pp. 75809–75821, 2020.

[24] M. Langhammer and B. Pasca, "Efficient FPGA modular multiplication implementation," in *Proc. ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays*, Feb. 2021, pp. 217–223.

[25] K. Safiullah, J. Khalid, and S. Y. Ali, "High-speed FPGA implementation of full-word Montgomery multiplier for ECC applications," *Microprocessors Microsyst.*, vol. 62, pp. 91–101, Oct. 2018.

[26] L. Henzen and W. Fichtner, "FPGA parallel-pipelined AES-GCM core for 100G Ethernet applications," in *Proc. ESSCIRC*, Sep. 2010, pp. 202–205.

[27] C. Rebeiro and D. Mukhopadhyay, "Power attack resistant efficient FPGA architecture for Karatsuba multiplier," in *Proc. 21st Int. Conf. VLSI Design (VLSID)*, Jan. 2008, pp. 706–711.

[28] H. Fan, J.-G. Sun, M. Gu, and K.-Y. Lam, "Overlap-free Karatsuba–Ofman polynomial multiplication algorithms," *IET Inf. Secur.*, vol. 4, no. 1, pp. 8–14, Mar. 2010.

[29] J. Xie, P. K. Meher, M. Sun, Y. Li, B. Zeng, and Z.-H. Mao, "Efficient FPGA implementation of low-complexity systolic Karatsuba multiplier over GF(2^m) based on NIST polynomials," *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 64, no. 7, pp. 1815–1825, Jul. 2017.

[30] J. Samanta, R. Sultana, and J. Bhaumik, "FPGA based modified Karatsuba multiplier," in *Proc. Int. Conf. VLSI Signal Process. (ICVSP)*, vol. 10, 2014, p. 12.

[31] 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, 2018.

[32] Y. Li, Y. Zhang, and W. He, "Fast hybrid Karatsuba multiplier for type II pentanomials," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 28, no. 11, pp. 2459–2463, Nov. 2020.

[33] S. Sau, P. Baidya, R. Paul, and S. Mandal, "Binary field point multiplication implementation in FPGA hardware," in *Intelligent and Cloud Computing*. Singapore: Springer, 2021, pp. 387–394.

[34] V. R. Kolagatla, V. Desalphine, and D. Selvakumar, "Area-time scalable high radix Montgomery modular multiplier for large modulus," in *Proc. 25th Int. Symp. VLSI Design Test (VDAT)*, Sep. 2021, pp. 1–4.

[35] K. Gowreesrinivas and P. Samundiswary, "FPGA implementation of single-precision floating point multiplication with Karatsuba algorithm using vedic mathematics," in *Smart Computing and Informatics*. Singapore: Springer, 2018, pp. 515–524.

[36] P. L. Montgomery, "Five, six, and seven-term Karatsuba-like formulae," *IEEE Trans. Comput.*, vol. 54, no. 3, pp. 362–369, Mar. 2005.

[37] M. G. Find and R. Peralta, "Better circuits for binary polynomial multiplication," *IEEE Trans. Comput.*, vol. 68, no. 4, pp. 624–630, Apr. 2019.

[38] S. E. Mathe and L. Boppana, "Low-power and low-hardware bit-parallel polynomial basis systolic multiplier over GF(2^m) for irreducible polynomials," *ETRI J.*, vol. 39, no. 4, pp. 570–581, 2017.

[39] B. Rashidi, "Throughput/area efficient implementation of scalable polynomial basis multiplication," *J. Hardw. Syst. Secur.*, vol. 4, pp. 120–135, Jan. 2020.

[40] G. Zhou, H. Michalik, and L. Hinsenkamp, "Complexity analysis and efficient implementations of bit parallel finite field multipliers based on Karatsuba–Ofman algorithm on FPGAs," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 18, no. 7, pp. 1057–1066, Jul. 2010.

[41] J. P. S. Dhanaselvam and R. Jenifer, "Real time implementation of recursive Karatsuba algorithm using VLSI," *Int. J. Innov. Res. Comput. Commun. Eng.*, vol. 4, no. 2, pp. 161–168, 2016.

[42] J. von zur Gathen and J. Shokrollahi, "Efficient FPGA-based Karatsuba multipliers for polynomials over F_2," in *Proc. Int. Workshop Sel. Areas Cryptogr.* Berlin, Germany: Springer, 2005, pp. 359–369.