Robust Efficient And High-Speed Overlap-Free Karatsuba-Based Finite-Field Multiplier

R.Vaishnavi [1], E L Shajini[2]
[1, 2] Dept of ECE
[1, 2] Vins Christian College Of Engineering

Abstract- *Cryptography systems have become inseparable parts of almost every communication device. Among cryptography algorithms, public-key cryptography, and in particular elliptic curve cryptography (ECC), has become the most dominant protocol at this time. In ECC systems, polynomial multiplication is considered to be the most slow and area consuming operation. Here proposes a novel hardware architecture for efficient field-programmable gate array (FPGA) implementation of finite field multipliers for ECC using overlap-free karatsuba algorithm. Proposed system implemented on different for various operand sizes, and performance parameters were determined. In existing approximate recursive multipliers are low-power designs that exploit approximate building blocks to scale up to their final size. Here present two novel 4 x 4 approximate multipliers obtained by carry manipulation. They are used to compose 8 x 8 designs with different error-performance trade-off. Comparing to state-of-the art works, the proposed method resulted in a lower combinational delay and area–delay product indicating the efficiency of design.*

Keywords- Cryptography, public-key cryptography ,in particular elliptic curve cryptography, efficient field-programmable gate array, overlap-free karatsuba.

I. INTRODUCTION

Several applications, such as elliptic curve cryptography (ECC), error correcting codes, and signal processing, require finite field multiplications as key units in the main arithmetic operation-processes. For example, in, the National Institute of Standards and Technology (NIST) has recommended five binary extension fields, i.e., m = 163, 233, 283, 409, and 571, for the elliptic curve digital signature algorithm (ECDSA) implementation. Generally, ECC/ECDSA is based on a point multiplication kP, wherek is an integer and P is a point on the elliptic curve. The point multiplication involves two major operations, i.e., point addition and point doubling (these two operations can be realized by affine or projective coordinates). To avoid inversion operation, point addition and point doubling can employ the projective coordinates to have only operations of finite field addition, squaring, and multiplication (addition and squaring are simpler operations than the multiplication). Based on this consideration, finite field multiplication surely becomes the bottleneck of the point multiplication and has gained substantial attentions from the research community recently.

In the binary extension fields, multiplication based on different field representation such as polynomial basis (PB), dual basis (DB), and normal basis (NB) involves different arithmetic operations and hardware architectures. Generally, the structures of the finite field multiplications can be classified into three major categories: bit-serial, digit serial, and bit-parallel architectures. Among these structures, the digit-serial multipliers can provide tradeoff between space and time complexities and thus are seen as ideal candidate in practical implementation. Recent reports have shown that the digit-serial multipliers based fast algorithm can achieve lower complexity than the conventional strategy.

Beginning at 2015, Fan and Hasan have pointed out that the bit-parallel multiplication using Karatsuba algorithm (KA), Overlap-Free Karastsuba Algorithm (OFKA) and Toeplitz matrix-vector product (TMVP) decompositions, can lead to the sub-quadratic space complexity, which is suitable for the implementation of high-performance ECC. Recently, Lee and Meher and Lee et al. have derived new (a, b)-way KA and (a,b)-way Toeplitz block TMVP (TBTMVP) decompositions, respectively, to obtain subquadratic space complexity digit-serial multipliers.

Very-large-scale integration (VLSI) is the process of creating an integrated circuit by combining thousands of transistors into a single chip. VLSI began in the 1970s when complex semiconductor and communication technologies were being developed. VLSI is one of the basic building blocks of today's higher end or advance technology. VLSI is finding its applications in a variety of areas from simple electronic consumer products to very complex circuits used in space electronics. Nowadays the trend in digital hardware product is to integrate as much circuitry as possible onto a single-chip. VLSI has offered new opportunity to design the circuits which were not possible before with old circuits. Programmable logic devices (PLD) and Electronic Design

Automation (EDA) tools have added much advancement in VLSI design.

The microprocessor is a VLSI device. Before the introduction of VLSI technology most ICs had a limited set of functions they could perform. An electronic circuit consists of a CPU, ROM, RAM and other glue logic. Multiplication is the third basic mathematical operation of arithmetic, the others being addition, subtraction and division. The multiplication of integers, rational numbers, and real numbers is defined by a systematic generalization of this basic definition. Multiplication can also be visualized as counting objects arranged in a rectangle or as finding the area of a rectangle whose sides have given lengths. The area of a rectangle does not depend on which side is measured first, which illustrates the commutative property.

The Karatsuba algorithm (KA) for multiplying two polynomials was introduced. It saves coefficient multiplications at the cost of extra additions compared to the ordinary multiplication method. The basic KA is performed as follows. Consider two degree-1 polynomials A(x) and B(x) with n = 2 coefficients:

$$A(x) = a_1 x + a_0$$
$$B(x) = b_1 x + b_0$$

Let D0, D1, D0,1 be auxiliary variables with

$$D_0 = a_0 b_0$$
$$D_1 = a_1 b_1$$
$$D_{0,1} = (a_0 + a_1)(b_0 + b_1)$$

Then the polynomial C(x) = A(x)B(x) can be calculated in the following way:

$$C(x) = D_1 x^2 + (D_{0,1} - D_0 - D_1)x + D_0$$

This method requires three multiplications and four additions. The ordinary method requires n^2 multiplications and $(n-1)^2$ additions, i.e., four multiplications and one addition. Clearly, the KA can also be used to multiply integer numbers. The KA can be generalized for polynomials of arbitrary degree. The following algorithm describes a method to multiply two arbitrary polynomials with n coefficients using the one-iteration KA.

In the present section we explore various techniques to implement the 233 bit Karatsuba multiplier. We explore techniques like Padding, Binary, Simple, Generalized and the proposed Hybrid Karatsuba Multiplier. The design tradeoffs involved in the various architectures are reported.

The Padded Karatsuba multiplier is the most simple method of implementing a fully recursive Karatsuba multiplier for a field (2n) GF , where n= 2k + d and k is the largest integer such that 2k < n . The Padded Karatsuba multiplier extends the n bit multiplicands to 2k+1 bits by padding its most significant bits with 2k+1-n zeroes. This then allows the use of the basic recursive Karatsuba algorithm. The obvious drawback of this method is the extra arithmetic introduced due to the padding.

The Binary Karatsuba multiplier was proposed. The algorithm modifies the basic Karatsuba multiplier to handle any field of the form (2n) GF , where n = 2k + d, and k is the largest integer such that 2k < n . The algorithm splits each multiplicand into two terms, the higher term containing d bits and the lower term containing 2k bits. The higher term partial product (Ak Bh) is determined by a Binary Karatsuba algorithm for d bits. The number of times the Binary Karatsuba algorithm is called recursively depends on the hamming weight of n . For example, the binary equivalent of 233 is (11101001)2, therefore the Binary Karatsuba algorithm is used recursively for 5 iterations.

The Simple Karatsuba multiplier is the basic recursive Karatsuba multiplier with a small modification. If an n bit multiplication is needed to be done, n being any integer, it is split into two polynomials. The Al and Bl polynomials have n / 2 terms while the Ah and Bh polynomials have n / 2 terms. The Karatsuba multiplication can then be done with two n / 2 bit multiplications and a single n / 2 bit multiplication.

The basic Karatsuba multiplier defines a method to multiply two n bit polynomials using three n / 2 bit multipliers. This is achieved by splitting the n bit polynomial into a 2-term polynomial with each term having n / 2 bits. In it was shown that if A and B are two n k = 3 bit polynomials, the Karatsuba multiplier for 3-term polynomials can be used. This results in six multiplications and 13 additions.

II. LITERATURE SURVEY

Approximate computing has become an emerging research topic for energy-efficient design of circuits and systems. Many approximate arithmetic circuits have been proposed; therefore it is critical to summarize the available approximation techniques to improve performance and energy efficiency at a acceptable accuracy loss. This paper presents an overview of circuit-level techniques used for approximate arithmetic. This paper provides a detailed review of circuit-level approximation techniques for the arithmetic data path. Its focus is on identifying critical circuit-level approximation

techniques that apply to computational units and blocks. Approximate adders, multipliers, dividers, and squarer are introduced and classified according to their approximation methods. FFT and MAC are discussed as computational blocks that employ an approximate algorithm for implementation.

Improving the accuracy of a neural network (NN) usually requires using larger hardware that consumes more energy. However, the error tolerance of NNs and their applications allow approximate computing techniques to be applied to reduce implementation costs. Given that multiplication is the most resource-intensive and power-hungry operation in NNs, more economical approximate multipliers (AMs) can significantly reduce hardware costs. In this article, we show that using AMs can also improve the NN accuracy by introducing noise. We consider two categories of AMs: 1) deliberately designed and 2) Cartesian genetic programing (CGP)-based AMs. The exact multipliers in two representative NNs, a multilayer perceptron (MLP) and a convolutional NN (CNN), are replaced with approximate designs to evaluate their effect on the classification accuracy of the Mixed National Institute of Standards and Technology (MNIST) and Street View House Numbers (SVHN) data sets, respectively. Interestingly, up to 0.63% improvement in the classification accuracy is achieved with reductions of 71.45% and 61.55% in the energy consumption and area, respectively. Finally, the features in an AM are identified that tend to make one design outperform others with respect to NN accuracy. Those features are then used to train a predictor that indicates how well an AM is likely to work in an NN.

Approximate computing has emerged as a new paradigm for high-performance and energy-efficient design of circuits and systems. For the many approximate arithmetic circuits proposed, it has become critical to understand a design or approximation technique for a specific application to improve performance and energy efficiency with a minimal loss in accuracy. This article aims to provide a comprehensive survey and a comparative evaluation of recently developed approximate arithmetic circuits under different design constraints. Specifically, approximate adders, multipliers, and dividers are synthesized and characterized under optimizations for performance and area. The error and circuit characteristics are then generalized for different classes of designs. The applications of these circuits in image processing and deep neural networks indicate that the circuits with lower error rates or error biases perform better in simple computations, such as the sum of products, whereas more complex accumulative computations that involve multiple matrix multiplications and convolutions are vulnerable to single-sided errors that lead to a large error bias in the computed result. Such complex computations are more sensitive to errors in addition than those in multiplication, so a larger approximation can be tolerated in multipliers than in adders. The use of approximate arithmetic circuits can improve the quality of image processing and deep learning in addition to the benefits in performance and power consumption for these applications.

Multiplication is an essential image processing operation commonly implemented in hardware DSP cores. To improve DSP cores' area, speed, or energy efficiency, we can approximate multiplication. We present an approximate multiplier that generates two partial products using hybrid radix-4 and logarithmic encoding of the input operands. It uses the exact radix-4 encoding to generate the partial product from the three most significant bits and the logarithmic approximation with mantissa trimming to approximate the partial product from the remaining least-significant bits. The proposed multiplier fills the gap between highly accurate approximate non-logarithmic multipliers with a complex design and less accurate approximate logarithmic multipliers with a more straightforward design. We evaluated the multiplier's efficiency in terms of error, energy (power-delay-product) and area utilisation using NanGate 45 nm. The experimental results show that the proposed multiplier exhibits good area utilisation and energy consumption and behaves well in image processing applications.

Approximate multipliers are used in error-tolerant applications, sacrificing the accuracy of results to minimize power or delay. In this paper we investigate approximate multipliers using static segmentation. In these circuits a set of m contiguous bits (a segment of m bits) is extracted from each of the two n -bits operand, the two segments are in input to a small m×m internal multiplier whose output is suitably shifted to obtain the result. We investigate both signed and unsigned multipliers, and for the latter we propose a new segmentation approach. We also present simple and effective correction techniques that can significantly reduce the approximation error with reduced hardware costs. We perform a detailed comparison with previously proposed approximate multipliers, considering a hardware implementation in 28 nm technology. The comparison shows that static segmented multipliers with the proposed correction technique have the desirable characteristic of being on (or close to) the Pareto-optimal frontier for both power vs normalized mean error distance and power vs mean relative error distance trade-off plots. These multipliers, therefore, are promising candidates for applications where their error performance is acceptable. This is confirmed by the results obtained for image processing and image classification applications.

Libraries of approximate circuits are composed of fully characterized digital circuits that can be used as building blocks of energy-efficient implementations of hardware accelerators. They can be employed not only to speed up the accelerator development but also to analyze how an accelerator responds to introducing various approximate operations. In this paper, we present a methodology that automatically builds comprehensive libraries of approximate circuits with desired properties. Target approximate circuits are generated using Cartesian genetic programming. In addition to extending the EvoApprox8b library that contains common approximate arithmetic circuits, we show how to generate more specific approximate circuits; in particular, MxN-bit approximate multipliers that exhibit promising results when deployed in convolutional neural networks. By means of the evolved approximate multipliers, we perform a detailed error resilience analysis of five different ResNet networks. We identify the convolutional layers that are good candidates for adopting the approximate multipliers and suggest particular approximate multipliers whose application can lead to the best trade-offs between the classification accuracy and energy requirements. Experiments are reported for CIFAR-10 and CIFAR-100 data sets.

Approximate arithmetic circuits are an attractive alternative to accurate arithmetic circuits because they have significantly reduced delay, area, and power, albeit at the cost of some loss in accuracy. By keeping errors due to approximate computation within acceptable limits, approximate arithmetic circuits can be used for various practical applications such as digital signal processing, digital filtering, low power graphics processing, neuromorphic computing, hardware realization of neural networks for artificial intelligence and machine learning etc. The degree of approximation that can be incorporated into an approximate arithmetic circuit tends to vary depending on the error resiliency of the target application. Given this, the manual coding of approximate arithmetic circuits corresponding to different degrees of approximation in a hardware description language (HDL) may be a cumbersome and a time-consuming process—more so when the circuit is big. Therefore, a software tool that can automatically generate approximate arithmetic circuits of any size corresponding to a desired accuracy would not only aid the design flow but also help to improve a designer's productivity by speeding up the circuit/system development. In this context, this paper presents 'Approximator', which is a software tool developed to automatically generate approximate arithmetic circuits based on a user's specification. Approximator can automatically generate Verilog HDL codes of approximate adders and multipliers of any size based on the novel approximate arithmetic circuit architectures proposed by us. The Verilog HDL codes output by Approximator can be used for synthesis in an FPGA or ASIC (standard cell based) design environment. Additionally, the tool can perform error and accuracy analyses of approximate arithmetic circuits. The salient features of the tool are illustrated through some example screenshots captured during different stages of the tool use.

In approximate operators, dynamic truncation allows trading off energy and quality of computation at runtime. Although it exploits the specificity of the data being processed, its significant energy overhead over simple static truncation fundamentally limits its energy benefits. This brief describes a simple and efficient design methodology that reduces the energy consumption of dynamically truncated multipliers, based on a smart mapping of the partial products. A configurable hardware correction strategy is also proposed to enable graceful quality degradation, as well as more aggressive energy reduction at a given quality. When applied to Wallace multipliers, the proposed approach achieves quality, in terms of Mean Error Distance, up to 11× higher than the conventional dynamic truncation, at the same energy. In the case study of Discrete Cosine Transform compression, the proposed approximate multiplier reaches image qualities by 15-35% better, compared to prior art.

High speed multimedia applications have paved way for a whole new area in high speed error-tolerant circuits with approximate computing. These applications deliver high performance at the cost of reduction in accuracy. Furthermore, such implementations reduce the complexity of the system architecture, delay and power consumption. This paper explores and proposes the design and analysis of two approximate compressors with reduced area, delay and power with comparable accuracy when compared with the existing architectures. The proposed designs are implemented using 45 nm CMOS technology and efficiency of the proposed designs have been extensively verified and projected on scales of area, delay, power, Power Delay Product (PDP), Error Rate (ER), Error Distance (ED), and Accurate Output Count (AOC). The proposed approximate 4 : 2 compressor shows 56.80% reduction in area, 57.20% reduction in power, and 73.30% reduction in delay compared to an accurate 4 : 2 compressor. The proposed compressors are utilised to implement 8 × 8 and 16 × 16 Dadda multipliers. These multipliers have comparable accuracy when compared with state-of-the-art approximate multipliers. The analysis is further extended to project the application of the proposed design in error resilient applications like image smoothing and multiplication.

Approximate multipliers attract a large interest in the scientific literature that proposes several circuits built with

approximate 4-2 compressors. Due to the large number of proposed solutions, the designer who wishes to use an approximate 4-2 compressor is faced with the problem of selecting the right topology. In this paper, we present a comprehensive survey and comparison of approximate 4-2 compressors previously proposed in literature. We present also a novel approximate compressor, so that a total of twelve different approximate 4-2 compressors are analyzed. The investigated circuits are employed to design 8 × 8 and 16 × 16 multipliers, implemented in 28nm CMOS technology. For each operand size we analyze two multiplier configurations, with different levels of approximations, both signed and unsigned. Our study highlights that there is no unique winning approximate compressor topology since the best solution depends on the required precision, on the signedness of the multiplier and on the considered error metric.

III. PROPOSED SYSTEM

The conventional multiplication method is not the most efficient method, other methods, such as KA and its variations, have been developed.

For two polynomials of degree one $(n = 2), A(x) = a_1 x + a_0$and $B(x) = b_1 x + b_0$, DFG for the 4-bit multiplier is represented in 2(b). It is worth noting that the 4-bit multiplier is constructed recursively using the 2-bit submultipliers block I includes splitting, submultiplication, and alignment stages. Furthermore, block II calculates the overlaps of common terms. For an n-bit multiplier, consider two n-term polynomials. $A(x)$ and $B(x)$, which are in GF(2^n). These polynomials with n − 1 degree are presented as follows:

$$A(x) = \sum_{i=0}^{n-1} a_i x^i$$

$$B(x) = \sum_{i=0}^{n-1} b_i x^i$$

where a_i and b_i are polynomial coefficients.

$A(x)$ and $B(x)$ split into most (A_H, B_H) and least (A_L, B_L) significant halves as follows:

$$A(x) = x^m \sum_{i=0}^{m-1} a_{m+i} x^i + \sum_{i=0}^{m-1} a_m x^i = A_H x^m + A_L$$

$$B(x) = x^m \sum_{i=0}^{m-1} b_{m+i} x^i + \sum_{i=0}^{m-1} b_m x^i = B_H x^m + B_L$$

where n = 2m.Using KA, the product of $A(x)\ B(x)$ could be calculated recursively as

$$A(x)B(x) = (A_H x^m + A_L)(B_H x^m + B_L)$$
$$A(x)B(x) = P_2(x)x^{2m} + [P_1(x) - P_2(x) - P_0(x)]x^m + P_0(x)$$

Where

$$P_2 = A_H B_H$$
$$P_1 = (A_H + A_L)(B_H + B_L)$$
$$P_0 = A_L B_L$$

Three submultipliers are required in order to obtain the multiplication results using KA. In terms of complexity analysis, the number of gates required for implementation of an n-bit multiplier is

$KA_{XOR}(n) = 3\ KA_{XOR}(n/2) + 4n - 4$
$KA_{AND}(n) = 3\ KA_{AND}\ (n/2)$
$T_{KA}(n) = 3\ Tx + T_{KA}(n/2)$.

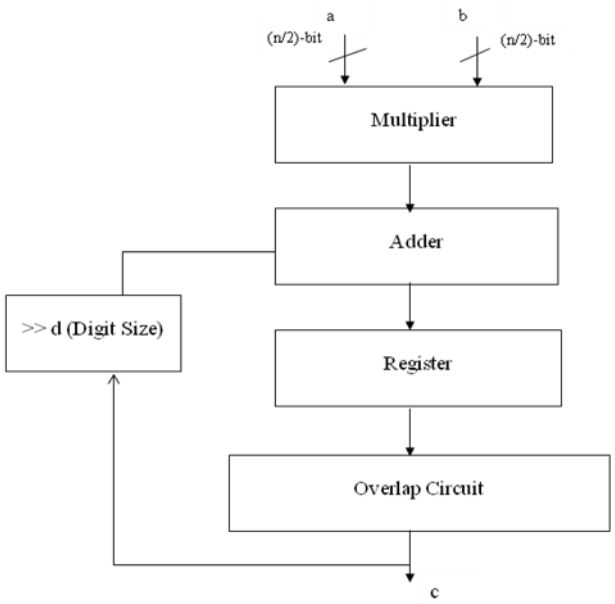

Fig.3.1 Proposed overlap-free Karatsuba-based multiplication strategy

The nonrecursive forms of these equations are as follows:
$KA_{XOR}(n) = 6\ n^{\log2(3)} - 8n + 2$
$KA_{AND}(n) = n^{\log2(3)}$
$T_{KA}(n) = Ta + (\ 3\ \log2(n) - 1\ )\ T_x$

OVERLAP-FREE KARATSUBA ALGORITHM

The OKA is a speed-optimized version of the original Karatsuba. In this method, to improve the longest path delay, inputs are split into odd and even orders instead of the high

and low parts. Once more, it is assumed that $A_{(x)}$ and $B_{(x)}$ are two polynomial in G $F^{(2n)}$ and n = 2m. These polynomials could be rewritten as

$$A(x) = x^m \sum_{i=0}^{m-1} a_{2i}\, x^{2i} + \sum_{i=0}^{m-1} a_{2i+1}\, x^{2i+1}$$

$$B(x) = x^m \sum_{i=0}^{m-1} b_{2i}\, x^{2i} + \sum_{i=0}^{m-1} b_{2i+1}\, x^{2i+1}$$

$$A(x) = x^m \sum_{i=0}^{m-1} a_{2i}\, y^i + x \sum_{i=0}^{m-1} a_{2i+1}\, y^i = A_e(y) + x A_e(y)$$

$$B(x) = x^m \sum_{i=0}^{m-1} b_{2i}\, y^i + x \sum_{i=0}^{m-1} b_{2i+1}\, y^i = B_e(y) + x B_e(y)$$

where A_e and B_e include the even order and A_o and B_o include the odd-order terms of polynomials $A_{(X)}$ and $B_{(x)}$, respectively.

Following a similar approach to the KA, the polynomial multiplication could be calculated as: $A_{(x)}B_{(x)} = (A_e(y) + x\ A_o(y)) \times (B_e(y) + x\ B_o(y)) = G_2(y)y + [G_1(y) - G_2(y) - G_0(y)]x + G_0(y)$
Where

$G_0 = A_eB_e$
$G_1 = (A_o + A_e)(B_o + B_e)$
$G_2 = A_oB_o$

Similar to the KA, the overlap-free algorithm also uses three sub multipliers. However, term $G_0(y) + yG_2(y)$ has only odd exponents (x^{2n+1}), and term $G_1(y)$ contains only terms with even exponents (x^{2n}). Therefore, there is no overlap between the components of these two terms, which allows removing an XOR gate from the critical path of the Karatsuba multiplier. It should be noted that the DFG for a first-order OKA multiplier is similar to that of the KA, for the 4-bit multiplier, the critical path of overlap-free is one XOR gate delay shorter than a Karatsuba multiplier. By solving the recursive equation for area and space requirements, the estimated values of the OKA implementation are as follows:

$OKA_{XOR}(n) = 3\ OKA_{XOR}(n/2) + 4n - 4$
$OKA_{AND}(n) = 3\ OKA_{AND}(n/2)$
$T_{OKA}(n) = 2\ T_x + T_{OKA}(n/2)$
$OKA_{XOR}(n) = 6\ n^{\log2(3)} - 8n + 2$
$OKA_{AND}(n) = n^{\log2(3)}$
$T_{OKA}(n) = Ta + (2\ \log2(n) - 1)\ T_x$

Overlap-free Karatsuba space complexity is the same as the KA. However, its time complexity is decreased from $(3\ \log_2(n) - 1)T_x$ in KA to $(2\ \log_2(n) - 1)T_x$ in OKA.

PROPOSED MULTIPLICATION STRATEGY

A new and efficient implementation of the finite-field multiplier is proposed. The proposed implementation strategy is obtained by studying the theoretical boundaries of area and delay of conventional, Karatsuba, and overlapfree methods. An observed trend is used as a guideline for creating finite-field multipliers of various sizes. Moreover, hardware resources requirements and combinational delay for two different implementation approaches, theoretical gatebased analysis and FPGA, are evaluated. The total number of gates for hardware implementation of the binary polynomial multiplication algorithms for different operand sizes is presented . It can be observed that considering small operand sizes, the number of gates required for implementing CAs is lower than that of the KA. However, as the operand size grows, the number of gates for implementing CA becomes substantially higher than Karatsuba and overlap-free. As an example, for operand size of 409 bits, the CA requires almost 163% more gates than the Karatsuba or overlap-free.

The total combination delay in terms of gate delays for all three algorithms. It was assumed that the delay of the XOR and AND gates is the same; $T_x = T_a = T_g$. In this figure, the CA has the minimum delay; however, as operand size grows, the delay of conventional and overlapfree Karatsuba converge to almost the same value. On the other hand, Krartsuba's delay rises at a faster rate compared to the other two algorithms. It is also worth noting that the delays of 163, 193, and 233 bits recursive multipliers are the same since they have an equal number of stages.

Another interesting point is that the number of stages required to implement recursive multipliers, including KA and OKA, increases logarithmically with operand size rather than linearly. As an instance, in the case of the 233 bit, the first four stages recursively perform multiplication down to 15-bit multipliers. However, performing a 15-bit multiplication requires another four stages. It is also worth noting that the overall delay of these multipliers is determined by their corresponding number of stages.

To compare the efficiency of algorithms, area–delay product (ADP) was calculated for all algorithms where, on average, overlap-free has the minimum and conventional has the highest ADP. Since our target platform is FPGAs, not digital gate-based devices, we investigated on-FPGA time and space analysis of these algorithms and validated the outcome by implementing algorithms. When it comes to the FPGA devices, the building blocks constructing most functions are lookup tables (LUTs) and not combinational gates. The LUTs are considered as universal gates where any function could be

represented. Therefore, in order to accurately estimate the complexity and delay analysis, these structures should be implemented on the FPGA using the LUTs. DFG for FPGA implementation of a 2-bit and a 4-bit binary polynomial multiplier using six-input LUTs. As shown in this figure, irrespective of the number of gates and the difference in the DFGs, the LUT-based implementations are similar. The difference between LUT implementation of these algorithms in terms of performance and the number of LUTs becomes distinct as the operand size increases.

Therefore, the theoretical estimations that are conventionally used to evaluate the efficiency of the algorithms cannot be simply extended to their actual FPGA implementation. There are techniques to estimate the number of LUTs and delay for FPGA implementation of combinational circuits. However, in a pragmatic approach, all algorithms mentioned above for various operand sizes were implemented on FPGA. In terms of area, all algorithms utilize almost the same number of LUTs when their operand sizes are small. However, the difference in the area grows nonlinearly as the operand sizes increase. For example, for a 283-bit conventional multiplier, FPGA utilization is almost 69% higher than that of the KA. For 409 bit, the number of LUTs is approximately twice the number required for the Karatsuba implementation. It is expected that gap becomes wider for larger operand sizes. The overlap-free implementation, however, utilizes a slightly higher number of LUTs comparing to Karatsuba. As an instance, a 409-bit overlap-free requires almost 2.7% more LUTs than the Karatsuba implementation. Regarding the combinational delay, the CA is roughly 44% faster than Karatsuba on average. This number for overlap-free is smaller and is approximately 36%. The delays of Karatsuba and overlap-free are relatively close for small operand sizes. However, the difference grows with the size of multiplier operands in a way that for a 409-bit multiplier, overlap-free is almost 14% faster.

In terms of delay, for theoretical gate-based analysis as well as FPGA implementation results, the conventional method is the fastest, followed by the overlap-free and subsequently Karatsuba. In addition, the delay of overlap-free is close to Karatsuba on FPGA, while in theoretical results, it is closer to the CA. Moreover, the ADP for all three algorithms was evaluated As results denote, the ADP for the conventional method is smaller than the other methods for operand sizes less than 409 bit. Numerically, a 283-bit multiplier using the conventional method is 14% more efficient than Karatsuba and 9% more efficient than the overlap-free method. These numbers for a 93-bit multiplier are 64% and 66% consecutively. The trend indicates that the CA is the most efficient method for smaller operand sizes. It is also worth noting that for operand sizes larger than 93 bits, overlap-free is more efficient than the KA. It is expected that the overlap-free remains the most efficient method for larger operand sizes. Since the efficiency continues to trend toward the overlap-free method for large operand sizes, a hybrid approach should be considered to implement finite-field multiplication. The DFG for the proposed overlap-free-based multiplication strategy (OBS) method is shown in Fig. 2.1. The highest level is based on the overlap-free. However, the conventional approach is used at the first level (level 1). In order to illustrate the proposed hybrid approach.For each multiplier, the level in which the conventional method was used changed from level 1 to level 4, using a CA at lower levels helps to reduce the number of LUTs required for the FPGA implementation of the multipliers.

IV. RESULT AND DISCUSSION

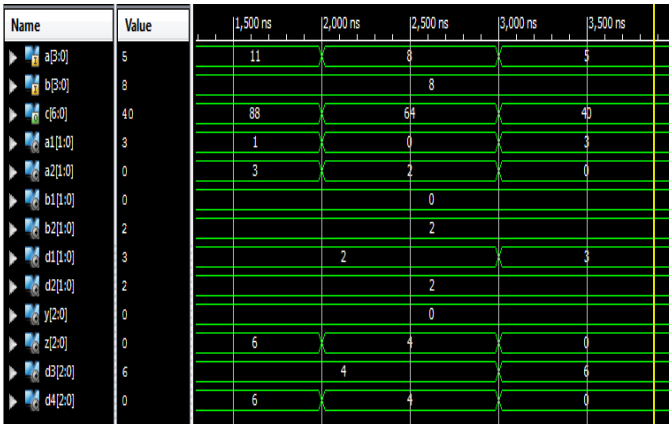

Figure 4.1 Proposed 4 bit Karatsuba multiplier

Device Utilization Summary				
Slice Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Registers	0	4,800	0%	
Number of Slice LUTs	13	2,400	1%	
Number used as logic	13	2,400	1%	
Number using O6 output only	6			
Number using O5 output only	0			
Number using O5 and O6	7			
Number used as ROM	0			
Number used as Memory	0	1,200	0%	
Number of occupied Slices	5	600	1%	
Number of LUT Flip Flop pairs used	13			
Number with an unused Flip Flop	13	13	100%	
Number with an unused LUT	0	13	0%	
Number of fully used LUT-FF pairs	0	13	0%	
Number of slice register sites lost to control set restrictions	0	4,800	0%	
Number of bonded IOBs	15	102	14%	
Number of RAMB16BWERs	0	12	0%	
Number of RAMB8BWERs	0	24	0%	
Number of BUFIO2/BUFIO2_2CLKs	0	32	0%	

Figure 4.2 Area analysis

Minimum period: No path found
Minimum input arrival time before clock: No path found
Maximum output required time after clock: No path found
Maximum combinational path delay: 7.725ns

Figure 4.3 Delay analysis

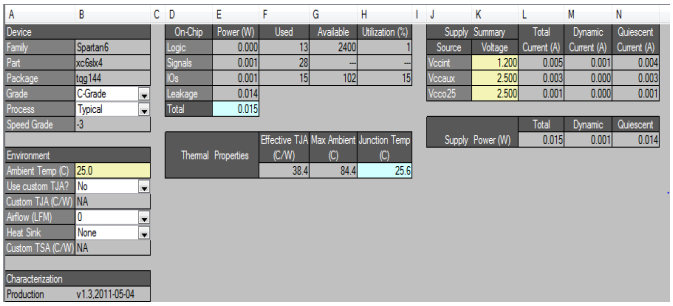

Figure 4.4 Power analysis

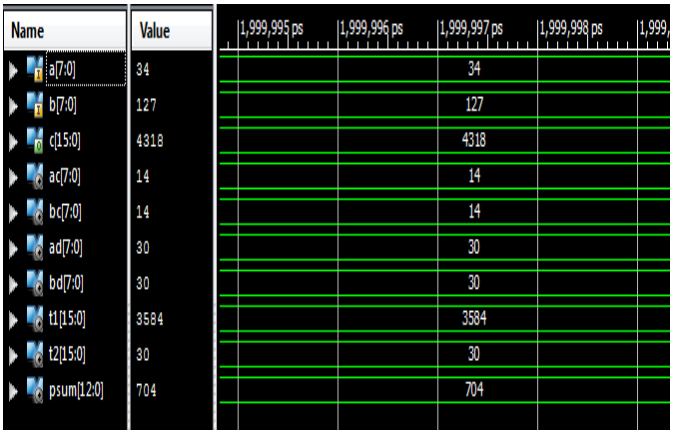

Figure 4.5 F Proposed 8 bit multiplier

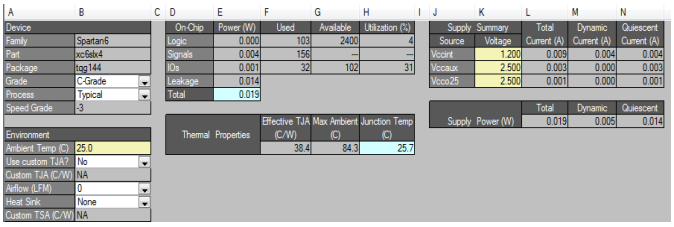

Figure 4.6 Power analysis

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: No path found
   Minimum input arrival time before clock: No path found
   Maximum output required time after clock: No path found
   Maximum combinational path delay: 14.754ns
```

Figure 4.7 Delay analysis

Device Utilization Summary

Slice Logic Utilization	Used	Available	Utilization	Not
Number of Slice Registers	7	4,800	1%	
Number used as Flip Flops	0			
Number used as Latches	0			
Number used as Latch-thrus	0			
Number used as AND/OR logics	7			
Number of Slice LUTs	103	2,400	4%	
Number used as logic	103	2,400	4%	
Number using O6 output only	64			
Number using O5 output only	0			
Number using O5 and O6	39			
Number used as ROM	0			
Number used as Memory	0	1,200	0%	
Number of occupied Slices	36	600	6%	
Number of LUT Flip Flop pairs used	103			
Number with an unused Flip Flop	96	103	93%	
Number with an unused LUT	0	103	0%	
Number of fully used LUT-FF pairs	7	103	6%	

Figure 4.8 Area analysis

V. CONCLUSION

Thus, the cryptography systems have become inseparable parts of almost every communication device. Among cryptography algorithms, public-key cryptography, and in particular elliptic curve cryptography (ECC), has become the most dominant protocol at this time. In ECC systems, polynomial multiplication is considered to be the most slow and area consuming operation. Proposed a novel hardware architecture for efficient field-programmable gate array (FPGA) implementation of finite field multipliers for ECC. Proposed system implemented on different for various operand sizes, and performance parameters were determined. Finally, 4 bit and 8 bit data for overlap free Karatsuba multiplier derived for ECC. They are used to compose 8 x 8 designs with different error-performance trade-off. Comparing to state-of-the art works, the proposed method resulted in a lower combinational delay and area–delay product indicating the efficiency of design.

REFERENCES

[1] Q. Xu, T. Mytkowicz, and N. S. Kim, "Approximate computing: A survey," IEEE Des. Test, vol. 33, no. 1, pp. 8–22, Feb. 2016.

[2] V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Analysis and characterization of inherent application resilience for approximate computing," in Proc. 50th ACM/EDAC/IEEE Des. Automat. Conf., 2013, pp. 1–9.

[3] K. Chen, P. Yin, W. Liu, and F. Lombardi, "A survey of approximate arithmetic circuits and blocks," Inf. Technol., vol. 64, no. 3, pp. 79–87, 2022.

[4] M. S. Ansari, V. Mrazek, B. F. Cockburn, L. Sekanina, Z. Vasicek, and J. Han, "Improving the accuracy and hardware efficiency of neural networks using approximate multipliers," IEEE Trans. Very Large Scale Integration (VLSI) Syst., vol. 28, no. 2, pp. 317–328, Feb. 2020.

[5] L. B. Soares, M. M. A. da Rosa, C. M. Diniz, E. A. C. da Costa, and S. Bampi, "Design methodology to explore hybrid approximate adders for energy-efficient image and video processing accelerators," IEEE Trans. Circuits Syst. I: Reg. Papers, vol. 66, no. 6, pp. 2137–2150, Jun. 2019.

[6] V. Gupta, D. Mohapatra, A. Raghunathan, and K. Roy, "Low-power digital signal processing using approximate adders," IEEE Trans. Comput.- Aided Des. Integr. Circuits Syst., vol. 32, no. 1, pp. 124–137, Jan. 2013.

[7] L. Chen, J. Han, W. Liu, P. Montuschi, and F. Lombardi, "Design, evaluation and application of approximate high-radix dividers," IEEE Trans. Multi-Scale Comput. Syst., vol. 4, no. 3, pp. 299–312, Jul.–Sep. 2018.

[8] M. Horowitz, "Computing's energy problem (and what we can do about it)," in Proc. IEEE Int. Solid-State Circuits Conf., 2014, pp. 10–14.

[9] H. Jiang, F. J. H. Santiago, H. Mo, L. Liu, and J. Han, "Approximate arithmetic circuits: A survey, characterization, and recent applications," Proc. IEEE, vol. 108, no. 12, pp. 2108–2135, Dec. 2020.

[10] W. Liu, J. Xu, D. Wang, C. Wang, P. Montuschi, and F. Lombardi, "Design and evaluation of approximate logarithmic multipliers for low power errortolerant applications," IEEE Trans. Circuits Syst. I: Reg. Papers, vol. 65, no. 9, pp. 2856–2868, Sep. 2018.

[11] M. S. Kim, A. A. D. Barrio, L. T. Oliveira, R. Hermida, and N. Bagherzadeh, "Efficient Mitchell's approximate log multipliers for convolutional neural networks," IEEE Trans. Comput., vol. 68, no. 5, pp. 660–675, May 2019.

[12] U. Lotric, R. Pilipovi c, and P. Buli c, "A hybrid radix-4 and approximate logarithmic multiplier for energy efficient image processing," Electron. Low-Size Low-Power Sensors Syst.: From Custom Des. Embedded Solutions, vol. 10, no. 10, May 2021, Art. no. 1175.

[13] A. G. M. Strollo, E. Napoli, D. De Caro, N. Petra, G. Saggese, and G. Di Meo, "Approximate multipliers using static segmentation: Error analysis and improvements," IEEE Trans. Circuits Syst. I: Reg. Papers, vol. 69, no. 6, pp. 2449–2462, Jun. 2022, doi: 10.1109/TCSI.2022.3152921.

[14] T. Yang, T. Ukezono, and T. Sato, "A low-power high-speed accuracycontrollable approximate multiplier design," in Proc. 23rd Asia South Pacific Des. Automat. Conf., 2018, pp. 605–610.

[15] M. Ce ska, J. Maty as, V. Mrazek, L. Sekanina, Z. Vasicek, and T. Vojnar, "ADAC: Automated design of approximate circuits," in Computer Aided Verification. Berlin, Germany: Springer, 2018.

[16] S. Ullah, S. S. Murthy, and A. Kumar, "SMApproxLib: Library of FPGAbased approximate multipliers," in Proc. 55th ACM/ESDA/IEEE Des. Automat. Conf., 2018.

[17] V. Mrazek, L. Sekanina, and Z. Vasicek, "Libraries of approximate circuits: Automated design and application in CNN accelerators," IEEE J. Emerg. Sel. Topics Circuits Syst., vol. 10, no. 4, pp. 406–418, Dec. 2020,

[18] P. Balasubramanian, R. Nayar, O. Min, and D. L. Maskell, "Approximator: A software tool for automatic generation of approximate arithmetic circuits," Computers, vol. 11, no. 1, Jan. 2022.

[19] S. Vahdat, M. Kamal, A. Afzali-Kusha, and M. Pedram, "TOSAM: An energy-efficient truncation- and rounding-based scalable approximate multiplier," IEEE Trans. Very Large Scale Integration (VLSI) Syst., vol. 27, no. 5, pp. 1161–1173, May 2019.

[20] F. Frustaci, S. Perri, P. Corsonello, and M. Alioto, "Approximate multipliers with dynamic truncation for energy reduction via graceful quality degradation," IEEE Trans. Circuits Syst. II: Exp. Briefs, vol. 67, no. 12, pp. 3427–3431, Dec. 2020.