Received March 27, 2020, accepted April 17, 2020, date of publication April 21, 2020, date of current version May 6, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.2989423*

Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

DAVIDE ZONI, ANDREA GALIMBERTI, AND WILLIAM FORNACIARI, (Senior Member, IEEE)



Dipartimento di Elettronica Informazione e Bioingegneria (DEIB), Politecnico di Milano, 20133 Milano, Italy

Corresponding author: Davide Zoni (davide.zoni@polimi.it)

This work was supported by the EU H2020 RECIPE Project under Grant 801137.

 **ABSTRACT** With the recent advances in quantum computing, code-based cryptography is foreseen to beone of the few mathematical solutions to design quantum resistant public-key cryptosystems. The binary polynomial multiplication dominates the computational time of the primitives in such cryptosystems, thus the design of ef cient multipliers is crucial to optimize the performance of post-quantum public-key crypto-graphic solutions. This manuscript presents a exible template architecture for the hardware implementation of large binary polynomial multipliers. The architecture combines the iterative application of the Karatsuba algorithm, to minimize the number of required partial products, with the Comba algorithm, used to optimize the schedule of their computations. In particular, the proposed multiplier architecture supports operands in the order of dozens of thousands of bits, and it offers a wide range of performance-resources trade-offs that is made independent from the size of the input operands. To demonstrate the effectiveness of our solution, we employed the nine con gurations of the LEDAcrypt public-key cryptosystem as representative use cases for large-degree binary polynomial multiplications. For each con guration we showed that our template architecture can deliver a performance-optimized multiplier implementation for each FPGA of the Xilinx Artix-7 mid-range family. The experimental validation performed by implementing our multiplier for all the LEDAcrypt con gurations on the Artix-7 12 and 200 FPGAs, i.e., the smallest and the largest devices of the Artix-7 family, demonstrated an average performance gain of 3.6x and 33.3x with respect to an optimized software implementation employing the *gf2x* C library.



 **INDEX TERMS** Computer arithmetic, FPGA, hardware design, multiplication, GF2, applied cryptography,post-quantum cryptography.



**I. INTRODUCTION**

Traditionally, the con dentiality provided by widely adopted public-key cryptosystems relies on the hardness of factoring large integers or computing discrete logarithms in a cyclic group. However, the recent advances in quantum computing pose a severe concern to the security of traditional public-key cryptographic schemes, since the employed computationally hard problems can be easily solved with a quantum computer. As a consequence, the design of quantum-computing resis-tant cryptographic primitives has gained importance lately, especially thanks to the U.S. National Institute of Standards and Technology (NIST) initiative, which aims at selecting a portfolio of primitives for standardization. In particular, the design of public-key cryptosystems that are resistant to

The associate editor coordinating the review of this manuscript and approving it for publication was Tony Thomas.

attacks performed with quantum computers imposes a change in the underlying computationally hard problem.

Code-based cryptography emerged as one of the few math-ematical post-quantum available techniques, together with lattice-based and hash-based cryptography [1]. Code-based cryptography leverages the hardness of decoding a syndrome obtained with a random linear block code and that of nding a xed weight code word in the said code [2]. Such problems belong to the NP-complete class and it is widely assumed that they cannot have a polynomial time solution even on quantum computers. McEliece proposed the rst cryptosystem relying on the hardness of the decoding problem [3], while several following proposals aimed at nding code families with ef - cient state-space representation to reduce the size of the cryp-tographic key-pairs. Quasi-cyclic low-density parity-check (QC-LDPC) [4] and quasi-cyclic moderate-density parity-check (QC-MDPC) [5] codes, employ circulant generator and parity check matrices for which all the rows are obtained by

|  |  |  |  |
| --- | --- | --- | --- |
| VOLUME 8, 2020 | This work is licensed under a Creative Commons Attribution 4.0 License. For more information, see https://creativecommons.org/licenses/by/4.0/ | 75809 |  |
|  |  |

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

cyclically rotating the rst one. The arithmetic of circulant matrices with size *p* can be proven to be isomorphic to the arithmetic of the polynomials modulo *xp* C 1 over the same eld as the coef cients of the circulant matrices. In particular, in the case of binary linear block codes, the arithmetic of *p p* circulant matrices over Z2 can be substituted by the arithmetic of polynomials in Z2[*x*]=(*xp* C 1), thus reducing signi cantly the size of the keys and achieving faster arithmetic operations.

We note that the multiplication operation dominates the computation for both McEliece, i.e., matrix multiplication, and QC-LDPC codes, i.e., polynomial multiplication. It is therefore crucial to provide an ef cient multiplication prim-itive for the performance of quantum-resistant public-key cryptosystems. In particular, available software solutions for different QC-LDPC codes involved in the NIST standardiza-tion process, e.g., LEDAcrypt [6] and BIKE [7], demonstrate the impossibility to cope with the required performance, espe-cially given the stiff increase in the key-size of the encryption schemes expected in the near future.

However, state-of-the-art hardware multipliers are meant to support computationally hard problems of traditional public-key cryptosystems, i.e., the factorization of large inte-gers and the computation of discrete logarithms in a cyclic group, thus they cannot be readily employed in code-based cryptosystems for two reasons. First, several proposals tar-get modular integer multiplication [8] [10], while code-based cryptography leverages the eld of binary polynomials. Second, the effectiveness of available binary polynomial mul-tipliers is strongly limited to the key-size of traditional public-key cryptosystems [11], i.e., few thousands of bits at most, thus they are not meant to ef ciently scale up to support the key-size of code-based cryptoschemes, i.e., dozens of thousands of bits.

*Contributions -* The manuscript presents an FPGA-oriented hardware design methodology to implement multi-pliers for large-degree binary polynomials. The possibility of employing the proposed design in post-quantum code-based cryptosystem implementations motivated its assess-ment against the Xilinx Artix-7 FPGA family, that is the sug-gested target technology for any hardware implementation within the NIST post-quantum cryptography competition. Our solution allows the designer to trade the resource utiliza-tion with the obtained performance in terms of throughput and latency, adding two relevant contributions with respect to the state-of-the-art:

Template architecture exibility - our template multi-plier architecture exposes three parameters to optimally trade, at design-time, performance and resource utiliza-tion on a wide range of FPGAs. First, the con gurable number of nested Karatsuba iterations allows to control the reduction of the number of partial products to be computed. Second, it is possible to con gure the number of partial products that are computed in parallel. Last, the internal bandwidth of the multiplier architecture is a design parameter as in state-of-the-art digit-serial multi-pliers.

75810

For each of the polynomial con gurations of the LEDAcrypt post-quantum public-key cryptosystem, performance results are collected considering the imple-mentation of the proposed template multiplier on differ-ent FPGAs.

Compared to an optimized software implementation, our solution showed an average performance speedup of 3.6x and 33.3x, by using the smallest (Artix-7 12) and the largest (Artix-7 200) FPGAs of the Xilinx Artix-7 family, respectively.

Operand size scalability - The proposed template multi-plier architecture allows to optimally select the resource-performance trade-off regardless of the size of the input operands. Such property is achieved by means of two design choices. First, the use of the FPGA block RAMs (BRAMs) instead of ip- ops, to store the operands, the result, and the intermediate values, allows to man-age operands ranging from few bits up to dozens of thousands of bits without altering the required memory storage. Second, the internal multiplier datapath and its primary input-output interface can be con gured to use different bandwidths.

Our exhaustive design space exploration demon-strates the possibility of implementing a performance-optimized multiplier, for each con guration of the LEDAcrypt cryptosystem, over the entire Xilinx Artix-7 family of mid-range FPGAs.

The rest of the manuscript is organized in four parts. Section [II](#page2) discusses the background on multiplication algo-rithms in the nite eld of binary polynomials and summa-rizes the state-of-the-art. The proposed multiplier design is discussed in Section [III.](#page5) Section [IV](#page8) presents the experimental results. Conclusions are drawn in Section [V.](#page12)

**II. BACKGROUND AND RELATED WORKS**

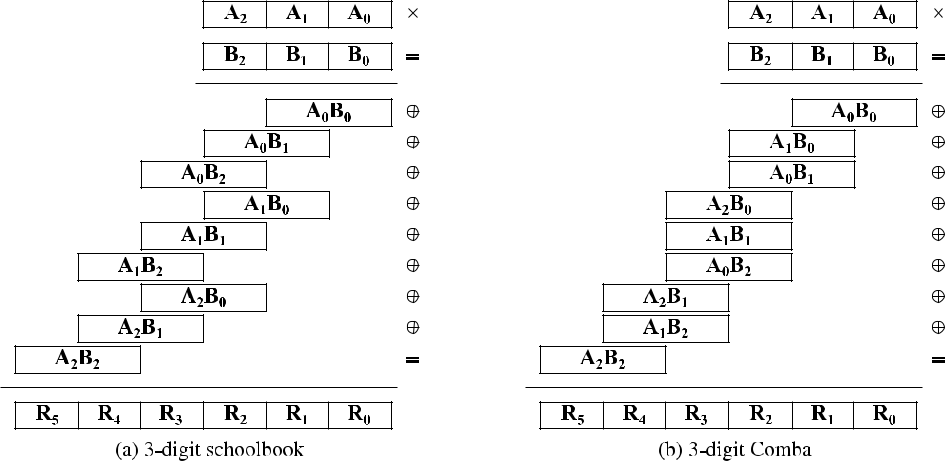
To support cryptographic computations, the state-of-the-art proposes several designs of ef cient multipliers implemented either in software or in hardware. The rest of this section is organized in two parts. Section [II-A](#page2) presents the theoretical background on the nite eld of binary polynomials used in code-based cryptosystems, as well as the multiplication methods adopted in the design of the proposed multiplier. Section [II-B](#page4) presents the state-of-the-art in terms of hardware and software implementations of the multipliers in such nite elds.

1. **BACKGROUND ON BINARY POLYNOMIAL MULTIPLICATION**

A nite eld, also called Galois eld, is a set that contains a nite number of elements on which the addition, subtraction, multiplication and division operations are de ned. Z2, or *GF*(2), is the Galois eld of two elements, i.e., the smallestGalois eld. The two elements of Z2 are usually referred to as 0 and 1, and they are respectively the additive and the multiplicative identities. The eld’s addition operation

VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials



**FIGURE 1.** Two multiplication methods implementing the long multiplication algorithm on digital systems.The number of partial products and additions grows up quadratically for both schoolbook and Comba algorithms. Comba offers a more efficient representation of intermediate results, and for this reason is able to optimize the memory write access pattern.

corresponds to the logical XOR operation, while the multipli-cation operation corresponds to the logical AND operation.

Polynomials with coef cients in Z2, i.e., 0 and 1, form a Galois eld, which is commonly referred to as Z2[*x*] or *GF*(2)[*x*]. The addition of two elements of such eld corresponds to a bitwise XOR. The multiplication, instead, consists in the multiplication of the two binary polynomials, followed by a reduction with respect to an irreducible poly-nomial, which is taken from the construction of the eld. For example, Z2[*x*]=(*xp* C 1) is the Galois eld of polynomials with coef cients in Z2 for which the irreducible polynomial is *xp* C 1, thus polynomials which belong to such eld have degree at most equal to *p* 1.

Multiplication in Z2[*x*] conceptually works like long mul-tiplication between integer numbers, except for the fact that the carry is always discarded instead of added to the more signi cant position. This property derives from the fact that the addition in Z2 corresponds to the logical XOR. For this reason, the multiplication operation in Z2[*x*] is also com-monly referred to as carry-less multiplication.

Considering the quasi-cyclic codes employed in many pro-posals for post-quantum code-based cryptosystems, the arith-metic of *p p* circulant matrices over Z2 can be substituted with the arithmetic of polynomials in Z2[*x*]=(*xp* C1). In code-based cryptosystems, matrix multiplication is the most com-putationally intensive operation of the encryption primitives. Since matrix multiplication corresponds to polynomial mul-tiplication when considering quasi-cyclic codes, it is crucial for the performance of these post-quantum cryptosystems to implement the latter operation in an effective way.

We will now discuss a few state-of-the-art algorithms to perform polynomial multiplication, i.e., the ones used in the proposed implementation. It is important to note that the multiplication algorithms have been selected to provide top-notch performance at reasonable complexity cost, according

VOLUME 8, 2020

to the range of sizes employed in quantum-resistant code-based cryptography. The use of more complex algorithms provides no extra performance but a non-negligible resource overhead, since they are expected to perform better when the operand sizes are orders of magnitude higher than what is needed to support code-based cryptography.

1) SCHOOLBOOK MULTIPLICATION

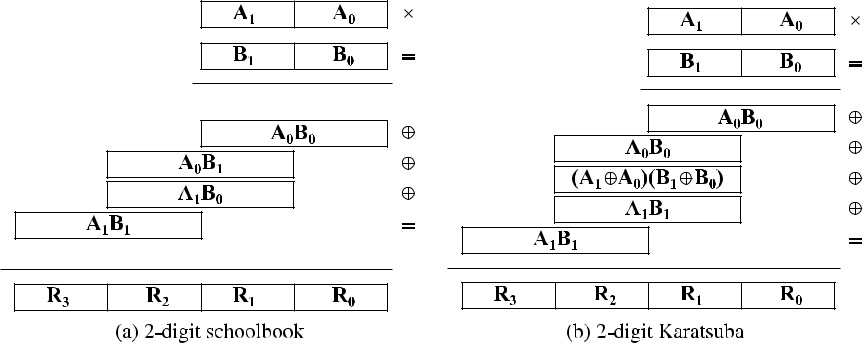
The schoolbook multiplication method implements the long multiplication for the execution on a digital system. Starting from the binary representation of the Z2[*x*] polynomials, each factor is split in digits according to the actual operand size of the digital system, e.g., 32- or 64-bits on current general purpose computers. The long multiplication algorithm is then implemented considering the digits as elementary units in the multiplication algorithm. Figure 1a depicts the schoolbook multiplication between two polynomials A and B. Each poly-nomial has been split in three digits, where their size is not explicitly reported since the method works for any possible size. We note that the bigger the digit size, the smaller the number of digits for each polynomial and the number of corresponding partial products to be computed. In particular, the number of the partial products and of the additions grows up quadratically with the number of digits. The larger is the digit size, the faster is the multiplication.

2) COMBA MULTIPLICATION

The Comba multiplication method [12] minimizes the num-ber of memory writes required by the schoolbook method by optimizing the order of computation of the partial prod-ucts (see Figure 1b). We note that the Comba algorithm requires exactly the same number of partial products and cor-responding additions as the schoolbook approach. However, the Comba algorithm minimizes the number of bits required to maintain in memory the sum of the partial products.

75811

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials



**FIGURE 2.** Multiplication of two-digit polynomials considering schoolbook and Karatsubaalgorithms. Karatsuba optimizes the computation by leveraging the intuition for which multiplications are more computationally expensive than additions in Z2[**x**]. In particular, schoolbook requires four multiplications and three additions, while Karatsuba performs the same computation using three multiplications and six additions.

For example, the nal value for the *R*0 digit is written in memory when the *A*0*B*0 partial product is ready. Moreover, the Comba method operates a right shift of digit size to trash the lower part of the *A*0*B*0 partial product, since it is not necessary anymore. In a similar manner, the nal value for the *R*1 digit is written in memory when the subsequent *A*0*B*1and *A*1*B*0have been computed. Again, the lower partof the intermediate result is trashed out since it is no longer useful. To this extent, the Comba method ensures a maximum of 2 *size*(*digit*) bits for the intermediate result, while the schoolbook algorithm requires at least *N size*(*digit*) bits, where *N* is the number of digits of each operand. In general, the optimized memory access pattern of the Comba solution, provides better performance than the schoolbook approach, even if the number of required multiplications and additions remains the same.

3) KARATSUBA MULTIPLICATION

The Karatsuba algorithm [13] optimizes the performance of the polynomial multiplication by reducing the number of partial products computations. The method leverages the intuition for which the multiplication is far more computa-tionally expensive than the addition in Z2[*x*]. Figure [2](#page4) depicts the multiplication of two operands, each of them split in two digits, using either the schoolbook (see Figure 2a) or the Karatsuba (see Figure 2b) approaches. The schoolbook solution requires four multiplications and three additions to perform the polynomial multiplication. In contrast, the Karatsuba approach requires three multiplications and six additions.

1. **RELATED WORKS ON BINARY POLYNOMIAL MULTIPLIERS**

The state-of-the-art contains several proposals implement-ing multipliers for the Galois eld of binary polynomi-als, either in the form of software libraries, hardware accelerators or custom extensions to the Instruction Set Architecture (ISA).

75812

1. SOFTWARE LIBRARIES AND INSTRUCTION SET ARCHITECTURE (ISA) SPECIAL INSTRUCTIONS

The *gf2x* [14] software library is the de-facto reference for fast multiplication of polynomials over Z2, implementing several multiplication algorithms to optimize the computa-tion for different operand sizes. In contrast, the *NTL* [15] library either implements only the Karatsuba multiplication algorithm, or it can act as an overlay to the *gf2x* library, while the *MPFQ* [16] library is speci cally tailored to deliver top-notch performance for nite- elds of moderate size, when the modulus size is known in advance.

From the Instruction Set Architecture (ISA) point of view, Intel introduced the *PCLMULQDQ* instruction and the corresponding hardware support in its Westmere architec-ture, to accelerate the computation of the AES in Galois Counter mode (AES-GCM) algorithm for authenticated encryption [17]. The PCLMULQDQ instruction performs the carry-less multiplication of two 64-bit operands, and the same operation is performed by the *VMULL.P64* instruc-tion on *ARM* targets [18]. The work in [19] leverages the *VPCLMULQDQ* instruction that will be supported in futureIntel Ice Lake solutions, and which is the vectorized exten-sion of PCLMULQDQ. In addition to its main intended use to further accelerate AES-GCM, the authors exploit it to compute large-degree binary polynomial multiplications, i.e., polynomials with degree greater than 511. In particular, results considering polynomials of degree up to 216 predict a 2x speed-up compared to current computing platforms.

1. HARDWARE ACCELERATORS

The state-of-the-art contains several architectures implement-ing ad-hoc hardware accelerators for the Galois eld of binary polynomials, either in the form of bit-serial, digit-serial, or bit-parallel multipliers. The bit-serial architectures have a low hardware complexity, thus they are well-suited for low-power and resource-constrained implementations. In particu-lar, such hardware accelerators output the *M*-bit result after

* clock cycles, thus their latency strictly depends on the size

VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

of the input. Reference [20] presents a low-power bit-serial multiplier architecture for binary polynomials for which an *M*-bit multiplier implementation requires 28 *M* gates. Thelimited performance and exibility in trading performance and area utilization of bit-serial architectures, prevents their use to design large multipliers, i.e., with operands of size in the order of dozens of thousands of bits.

In contrast, bit-parallel architectures are intended for performance-oriented implementations, since they perform the *M*-bit multiplication in one clock cycle. However, they are characterized by a high critical path delay and a high area consumption, which grows up more than linearly with the size of the operands [11]. To this end, the bit-parallel multipliers in the state-of-the-art are limited to relatively small operand sizes, i.e., one or two thousands of bits at the most. Reference

1. details the realization of the optimal bit-parallel design given the structure of the target binary polynomial Galois eld, i.e., the size of the polynomials of the eld and its associated irreducible polynomial.

We note that all bit-parallel solutions leverage the size of the operands to deliver ef cient ad-hoc architectures. To this end, each architecture is customized for a speci c Galois eld and it is therefore not reusable. The limited exibility and the hardware complexity that grows with the size of the operands make the bit-parallel architectures unsuitable to design large binary multipliers intended to be implemented on a large variety of FPGA devices, regardless of the size of the operands.

Differently from bit-parallel solutions, digit-serial polyno-mial basis multipliers offer a superior design exibility. In particular, the operands are organized in digits, i.e., chunks with a xed number of bits, and the multiplication pro-ceeds on a digit-by-digit basis. The possibility to con g-ure the size of the digit allows to trade the performance with the resource utilization. Reference [22] presents a low-area and scalable digit-serial architecture to perform poly-nomial basis multiplications over Z2[*x*]. Two digit-serial architectures for multiplication over Galois elds employing the normal basis representation are presented in [23], [24]. By rewriting the multiplication equations in a normal basis form, the design in [23] can reduce both the hardware com-plexity and the combinational critical path. In contrast, the digit-serial multiplier presented in [24] aims to speedup the exponentiation and the point multiplication, in any case a double multiplication is required and traditional schemes are performance-limited due to data dependences.

We note that the scalability offered by digit-serial solu-tions is limited to the possibility of con guring the size of the digit, i.e., the number of bits that are processed in parallel. Normally, state-of-the-art solutions are validated on limited operand sizes, less than few thousands of bits, thus the scalability issues of such solutions have not been fully highlighted. Differently, the implementation of large binary multipliers requires to extend the exibility of current digit-serial architectures with the use of fast multiplication algorithms to aggressively reduce the number of computed

VOLUME 8, 2020

partial products, without increasing the design complexity. In particular, several works in the state-of-the-art demonstrate the possibility of implementing the Karatsuba algorithm into the multiplier to minimize the number of computed partial product and, thus, to improve the overall multiplication per-formance. Reference [25] proposes a hardware multiplier employing an ad-hoc implementation of the Karatsuba algo-rithm for 240-bit polynomials. The design takes 30 clock cycles to perform a single multiplication, but the ad-hoc combinational logic structure severely thwarts the scalability of the multiplier. [26] presents a hardware multiplier relying on a Karatsuba-like approach. Depending on the operand size, the solution optimizes the performance by allowing to split the operands either into 4, 5, 6 or 7 blocks. However, the xed architecture limits the scalability of the solution in the exploitation of the resources available in large FPGAs. Moreover, the design has been validated against polynomials of degree up to 99. Reference [27] compares two imple-mentations of binary polynomial multipliers targeting the encryption function of LEDAcrypt. Depending on the actual number of coef cients set to 1 in one of the two polyno-mials, the paper discusses the possibility of using dense-to-sparse binary polynomial multipliers rather than traditional Karatsuba-like dense-to-dense architectures. In particular, the dense-to-dense multiplier implements a single iteration of the Karatsuba algorithm and either one serial or three parallel Comba multipliers to compute the three partial products. Such multiplier works at 100 MHz and the parallel and serial versions are provided as two separate implementations. In contrast, the template multiplier presented in this manuscript operates at 143 MHz, can recursively apply the Karatsuba algorithm a con gurable number of times, and the architec-ture is parametric in the use of either serial or parallel Comba multipliers.

**III. METHODOLOGY**

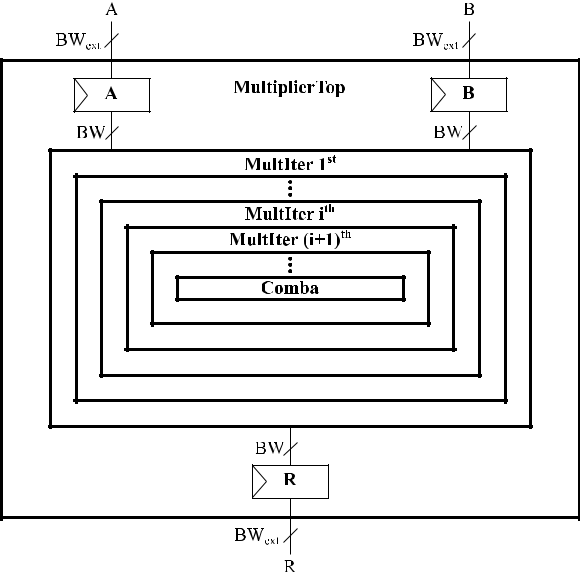
This section describes the microarchitecture proposed to implement a scalable and exible multiplier for large-degree binary polynomials. The design is meant to scale across a wide range of FPGAs rather than being hard-coded to a speci c FPGA. Moreover, its peculiar exibility permits to trade the performance with the utilization of the resources, depending on the actually selected FPGA. In particular, it is possible to implement multipliers for large binary polyno-mials even on resource constrained FPGA targets. Figure [3](#page6) depicts the architectural top view of the proposed multi-plier (MultiplierTop). The MultiplierTop module receives in input two operands, *A* and *B*, and outputs the result of the multiplication, *R*.

To ease the integration of the proposed component in real designs, the input and output interfaces offer a con gurable bandwidth, *BWext* , as well as input and output memory layers to store the inputs and the produced output, respectively. Such design completely decouples the bandwidth of the internal multiplier datapath (*BW* ) from the available external band-width (*BWext* ). In particular, the former has no externally

75813

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

**FIGURE 3.** Top view of the proposed multiplier template.



imposed constraints, while the latter can be constrained by the pin count or the data channel width of the System-on-Chip that integrates the multiplier. We note that the input and output memory layers are crucial components to oper-ate on large polynomials, since no physical interface can accommodate a datapath width of dozens of thousands of bits.

The architecture of the MultiplierTop module allows to implement a con gurable number of iterations of the Karatsuba algorithm, aggressively reducing the number of required partial products (see the MultIter blocks in Fig-ure [3)](#page6). At the end of the recursive application of the Karatsuba algorithm, the Comba multiplication algorithm performs the actual computation of the partial products (see Comba in Figure [3)](#page6). We note that the use of the Comba multiplication algorithm at the end of the Karatsuba iterations allows to optimally schedule the computation of each partial product, also considering that the size of the operands after the recur-sive application of the Karatsuba iterations is still too large to t into the combinational *BW BW* multiplier, which performs the carry-less multiplication between two BW-bit digits.

The rest of this section is organized in two parts. Section [III-A](#page6) details the architecture that allows to recursively apply the Karatsuba algorithm for a prede ned number of times. Such structure is meant to minimize the num-ber of required partial products and to maximize the level of parallelism to compute the remaining partial products. Section [III-B](#page7) discusses the architecture to actually compute the partial products. Depending on the required performance-resources trade-off, such con gurable computing architec-ture can implement either a single Comba multiplier, which computes the partial products in a serial way, i.e., one at a time, or a set of parallel Comba multipliers, which compute multiple partial products simultaneously.

75814

**A. KARATSUBA MULTIPLIER ARCHITECTURE**

The proposed architecture is based on a hybrid approach which leverages the recursive application of the Karatsuba algorithm, to minimize the number of partial products, and of the Comba algorithm, used as the leaf node of the recursion, to optimally schedule the operations to compute each partial product. Such design approach allows to separately optimize the modules implementing the Karatsuba and Comba algo-rithms.

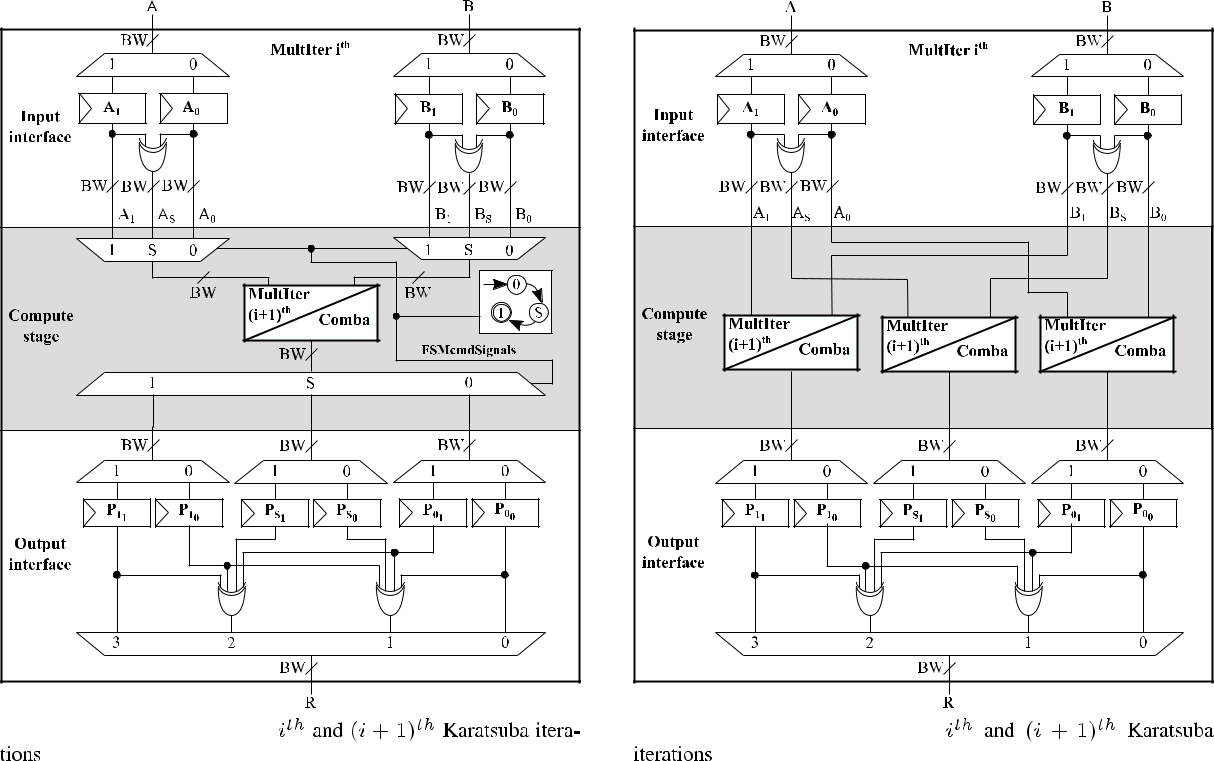
Figure [4](#page7) depicts the architecture of two nested Karatsuba iterations, *ith* and (*i* C 1)*th*, which is at the core of the iter-ative application of the Karatsuba algorithm. In particular, the inner Karatsuba iteration can implement either the serial (see Figure 4a) or the parallel (see Figure 4b) computation of the three partial products, thus allowing an additional level of exibility to trade the performance with the resource utilization.

Regardless of the serial or parallel implementation, each Karatsuba iteration (MultIter) receives two polynomials in input and it outputs the result of their carry-less multipli-cation. The input interface splits each one of the two poly-nomials in two halves, according to the Karatsuba algorithm. Each half of each polynomial, i.e., *A*1, *A*0, *B*1 and *B*0, is stored in a separate memory element. In a similar manner, the output interface delivers the nal multiplication result by composing the computed partial products according to the Karatsuba algorithm. We note that the proposed multiplier is parametric with respect to the implemented channel width, i.e., *BW* , that is used as an additional con guration option to trade performance with resource utilization. The compute stage receives the operands from the input interface and delivers the computed partial products to the output interface. The com-pute stage implements the logic to perform the computation of the three partial products required by the current Karatsuba iteration. We note that, instead of directly computing the three partial products by means of either one (serial) or three (parallel) Comba multipliers (see Comba in Figure 4a and Figure 4b), a nested application of the Karatsuba algorithm can be performed. In this scenario, the MultIter block represents the key element to implement the recursive appli-cation of the Karatsuba algorithm. In contrast, the Comba module represents the leaf node at the end of the recursive application of the Karatsuba algorithm.

From the architectural viewpoint, the use of either a par-allel or serial implementation of the compute stage repre-sents a con guration parameter of the proposed template multiplier. The parallel implementation of the compute stage only requires a proper connection of the input and the out-put signals to the nested MultIter/Comba modules (see Figure 4b). The serial implementation of the compute stage must orchestrate the computation of the three partial prod-ucts by leveraging the single, i.e., shared, computing block (MultIter/Comba) (see Figure 4a). To this purpose, a simple nite-state-machine drives the multiplexing infras-tructures to forward the correct part of the operands from the

VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials



**FIGURE 4.** The proposed architecture implements a configurable number of nested Karatsuba algorithm iterations. Each iteration can beimplemented as either serial, i.e., with one module that executes sequentially all the three Karatsuba partial products (see Figure 4a), or parallel, i.e., with three parallel submodules (see Figure 4b). The final iteration implements the Comba multiplication, as depicted in Figure [5.](#page7) **Ai** , **Bi** and **Pjk** are memories with a **BW** -bit bandwidth.

storage elements of the *ith* MultIter module to the single compute unit, i.e., (*i* C 1)*th* MultIter/Comba.

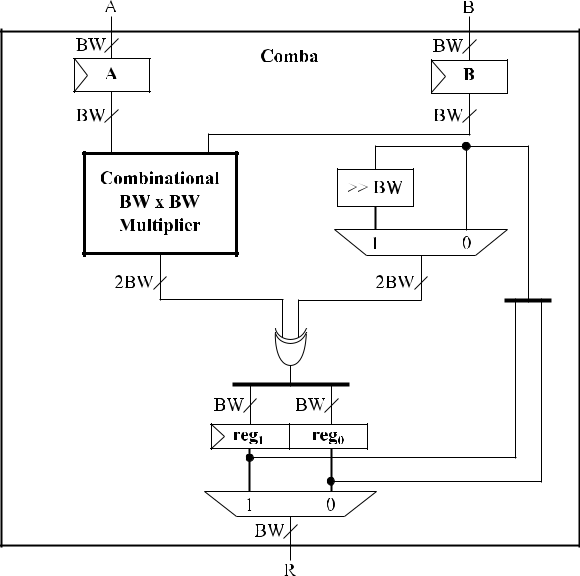
In summary, the proposed template multiplier architecture allows to exibly con gure *i)* the number of Karatsuba iter-ations to be implemented, *ii)* either the parallel or the serial computation for each of them, and *iii)* the internal channel width BW. The MultIter module implements an iteration of the Karatsuba algorithm, also offering the possibility to iterate the procedure by nesting parallel or serial instances of the same module.

**B. COMBA MULTIPLIER ARCHITECTURE**

Considering the proposed template multiplier architecture, the Comba multiplier (see Comba in Figure 4a and Figure 4b) is required to actually compute each partial product. To this end, the *Comba Multiplier* module represents the terminal block, i.e., the leaf node, of the recursive application of the Karatsuba algorithm.

Figure [5](#page7) depicts the architecture of the Comba module, which performs the multiplication of the input operands according to the schedule of the Comba algorithm [12]. We note that the iterative application of the Karatsuba algo-rithm minimizes the number of required partial products, while also halving the size of the operands at each iteration. However, the size of the operands in input to the Comba multiplier module is still in the order of thousands of bits,

**FIGURE 5.** The architecture of the proposed Comba multiplier. **A** and **B** are memories with a **BW** -bit bandwidth, **regi** are **BW** -bit registers.



thus far too large to perform a single combinational mul-tiplication. In contrast, the Comba multiplier assumes that each operand is made of a set of *BW* -bit digits and performs the multiplication, according to the Comba algorithm, in a digit-by-digit processing fashion. At the core of the Comba

|  |  |
| --- | --- |
| VOLUME 8, 2020 | 75815 |

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

**Algorithm 1** Bit-Level Combinational Multiplication.*A*and*B* Are *BW* -Bit Digits, *R* Is (2*BW* 1)-Bit Long. *A*[*i*], *B*[*i*] and *R*[*i*] Indicate Single Bits

1. **function** [*R*] CombinationalMul(*A*;*B*)

|  |  |
| --- | --- |
| 2: | **for** *i*20V*BW*1 **do** |
| 3: | **for** *j*20V*BW*1 **do** |

1. *R*[*i* C *j*]D *R*[*i* C *j*] (*A*[*i*] *B*[*j*])I
2. **end for**
3. **end for**
4. **end function**

module, the Combinational BW x BW Multiplier performs the multiplication between two digits (see Figure [5)](#page7). In particular, Algorithm [1](#page8) details the steps to perform the bit-level combinational multiplication of the two *BW* -bit digits according to the schoolbook multiplication algorithm.

The Comba multiplier schedules the *BW BW* multipli-cations according to the strategy proposed by Comba, i.e., producing a single *BW* -bit digit of the result at a time, by computing all the partial products contributing to it. This approach minimizes the number of bits required to maintain in memory the sum of the partial products. To implement this strategy, two *BW* -bit registers, *reg*1 and *reg*0, are employed to store the sum of all the contributions to the said portion of the result. *reg*1 and *reg*0 store respectively the *BW* most and least signi cant bits of the XOR of partial products computed by the combinational multiplier. When the computation of the sum is completed, the least signi cant *BW* bits, i.e., the *BW* bits stored in *reg*0, are committed to the output of the Comba Multiplier, while the most signi cant ones, i.e., the *BW* bits stored in *reg*1, are copied over in *reg*0.

**IV. EXPERIMENTAL EVALUATION**

This section discusses the results of the proposed hardware multiplier in terms of area, timing and performance (execu-tion time) with the nal goal of highlighting the exibility and the scalability of our solution. To demonstrate the exibility, we employed the LEDAcrypt public-key cryptosystem as our representative use case for large-degree binary polynomial multiplications.

To demonstrate the scalability, the proposed multiplier has been implemented on all the FPGAs of the mid-range Xilinx Artix-7 family, while results are only reported for the smallest and the largest FPGA in the family. We note that the Xilinx Artix-7 family offers the best performance per dollar and it has been suggested as the reference FPGA family target by NIST for its post-quantum cryptography competition. Perfor-mance results are compared with a state-of-the-art software implementation running on an Intel i7 processor.

The rest of this section is organized in three parts. Section [IV-A](#page8) overviews the LEDAcrypt cryptosystem, with emphasis on both the operand size and the computa-tional impact of the polynomial multiplication. Section [IV-B](#page9) details the experimental settings encompassing both hard-ware and software. Last, the experimental results in terms

75816

of resource utilization and performance are discussed in Section [IV-C.](#page10)

**A. LEDAcrypt CRYPTOSYSTEM**

We consider the Public-Key Cryptosystem (PKC) from the LEDAcrypt post-quantum cryptography suite [28] as a repre-sentative use case for a large-degree binary polynomial mul-tiplier. The LEDAcrypt PKC is a code-based cryptosystem that relies on the McEliece [3] cryptoscheme and employs a QC-LDPC code. LEDAcrypt is one of the nalists in the National Institute of Standards and Technology (NIST) ini-tiative for the standardization of quantum-resistant public-key cryptosystems.

The code underlying the PKC has code word length *pn*0 and information word length *p*(*n*0 1), where *n*02f2; 3; 4g and *p* is a large prime number. The LEDAcrypt PKC provides three security levels (equivalent respectively to AES-128, AES-192 and AES-256) and for each security level three different code rates, i.e., three different values of *n*0. To each of these nine con gurations corresponds a polynomial degree *p* (see Table [1)](#page8).

The encryption function of the LEDAcrypt PKC takes as its inputs a plaintext message composed as a 1 *p*(*n*01) binary vector *u*, an error vector composed as a 1 *pn*0 binary vector *e* and a public key *G* structured as (*n*0 1) *n*0 circulant blocks with size *p p*.

The ciphertext *c* is then computed as:

*c* D *u G* C *e*

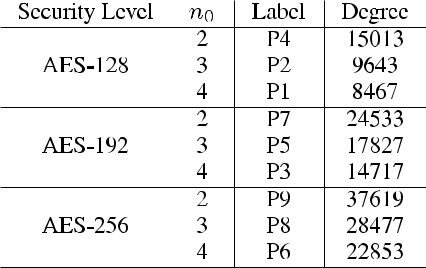
Because of the systematic form and the quasi-cyclic nature

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| of *G*, the *u G* matrix multiplication | corresponds to *n* | | | 0 | 1 |  |
|  | *p* |  | 1 |  |
| polynomial multiplications in Z2[*x*]=(*x* | |  | C 1). These *n*0 | |  |

multiplications are the most computationally expensive parts of the encryption algorithm, since the other operation is a computationally far simpler bitwise XOR.

The experimental results detailed in this section have been obtained for binary polynomials corresponding to each of the nine con gurations of the LEDAcrypt PKC, i.e., binary polynomial multiplications have been evaluated for polynomials of degree *p* for each con g-uration [29]. LEDAcrypt con gurations and their corre-sponding polynomial degrees are shown in more detail in Table [1.](#page8)

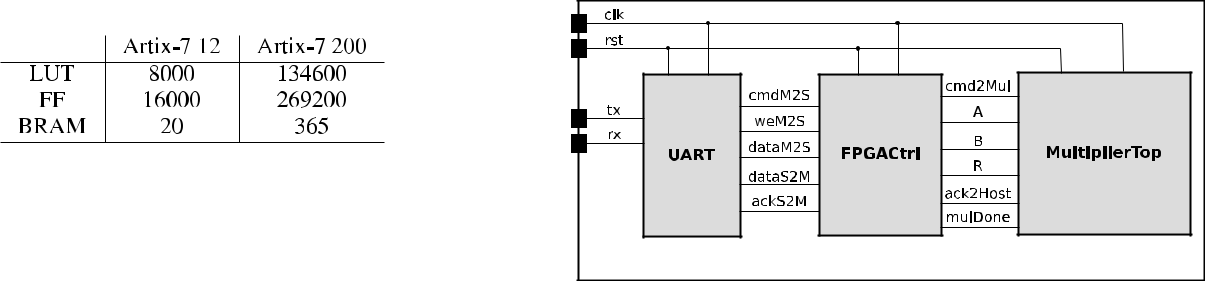
**TABLE 1.** Polynomial degrees of LEDAcrypt PKC configurations.



VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

**TABLE 2.** Available resources on Artix-7 12 and Artix-7 200 FPGAs.



* 1. **EXPERIMENTAL SETUP**

1. HARDWARE SETUP

The architecture for binary polynomial multiplication dis-cussed in Section [III](#page5) has been described in SystemVerilog and it has been implemented using the Xilinx Vivado 2018.2 hardware design suite. The experimental evaluation has been carried out on two FPGAs from the mid-range Xilinx Artix-7 family. The Artix-7 12 (*xc7a12tcsg325-1*) and the Artix-7 200 (*xc7a200tsbg484-1*) are respectively the lowest-end and the highest-end FPGAs of the Xilinx Artix-7 family (see details in Table [2)](#page9). Each design has been implemented con-sidering a 143 MHz operating frequency, i.e., a 7 ns clock period. It is worth noticing that for each considered FPGA we only reported the best multiplier con guration, i.e., the feasible one providing the best performance in terms of the time to compute the entire multiplication. Such con gura-tions have been identi ed after an extensive design space exploration which considered three parameters: *i)* the number of iterations of the Karatsuba algorithm, *ii)* the parallel or serial implementation of the compute stage for each iteration, and *iii)* the internal bandwidth of the multiplier. We note that the internal bandwidth, i.e., BW, has been xed at 64 bits, to match the bandwidth of the BRAM memories implemented in Xilinx Artix-7 FPGAs, thus delivering the optimal data transfer.

2) SOFTWARE SETUP

We used the *gf2x* C library [30] (version 1.3.0) as the software reference to compare the performance of the implemented hardware multipliers. The *gf2x* C library implements the Karatsuba, Toom-Cook and, for very large polynomials, FFT multiplication algorithms; it represents the state-of-the-art for software-implemented large binary polynomial multiplica-tions. To provide a representative comparison, we tuned the library according trough its automatic tuning procedure to select the most appropriate multiplication algorithm depend-ing on the underlying hardware and polynomial degree. The experimental evaluation of the software-implemented multi-plication has been carried out on an Intel Core i7-6700HQ processor forcing a xed operating frequency of 3.5 GHz to avoid performance variability due to the power management controller. For each value of *p* as de ned in the LEDAcrypt PKC use case, the execution time of the software reference was obtained by averaging 10000 execution times for a single multiplication. For each polynomial degree, measurements were taken only after a warm-up run of 1000 multiplications.

3) FUNCTIONAL VALIDATION

The functional validation is meant to check the correctness of the multiplication results obtained from the hardware

VOLUME 8, 2020

**FIGURE 6.** Hardware setup for the functional assessment of the proposed

binary multiplier. The hardware setup is made of three parts. The UART

module allows the communication between the host computer and the

Functional Validation Architecture. The FPGA controller

(FPGACtrl) coordinates the communication with the host computer and

the hardware execution of the binary multiplication. The multiplier

(MultiplierTop) implements the version of the proposed binary

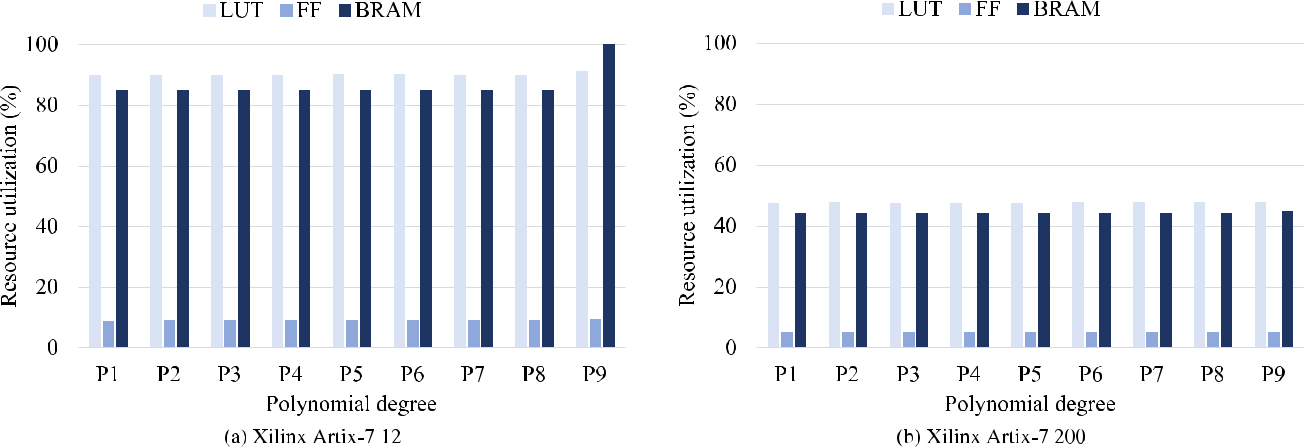
multiplier architecture optimized for the underlying FPGA .

implementation of our multiplier template architecture. We employed the *gf2x* C library [30] (version 1.3.0) as the ref-erence for our functional validation. In particular, for each polynomial degree de ned in the LEDAcrypt public-key cryptosystem, i.e. the 9 con gurations reported in Table [1,](#page8) we collected the results of the software execution of 10000 multi-plications where the input binary polynomials were randomly generated. The same multiplications have been computed through both the post-implementation (timing) simulation and the board prototype execution of different implementa-tions of our multiplier template architecture. For the post-implementation (timing) simulation, we implemented our multiplier targeting the Xilinx Artix-7 12 (*xc7a12tcsg325-1*) and the Xilinx Artix-7 200 (*xc7a200tsbg484-1*) FPGAs. For the board prototype execution, we implemented our multi-plier targeting the Digilent Nexys 4 DDR FPGA board, which features a Xilinx Artix-7 100 (*xc7a100tcsg324-1*) FPGA. In particular, we implemented a performance-optimized instance of our multiplier for each combination of the nine LEDAcrypt con gurations and the three considered FPGAs, i.e., Xilinx Artix-7 12, Xilinx Artix-7 100 and Xil-inx Artix7 200. Each one of the 27 implemented multipliers executes the 10000 multiplications and the output results are compared with the output of the corresponding software-executed multiplication.

A functional validation architecture (FVA) is employed to provide a uni ed validation infrastructure for both the post-implementation (timing) simulations and the board pro-totype executions (see Figure [6)](#page9). The FVA is made of three parts. The FPGA controller (FPGACtrl) communi-cates with the host computer to collect the input polynomials and to return the multiplication results. The UART mod-ule (UART) ensures a simple yet effective communication channel between the FPGA controller and the host computer. The MultiplierTop block represents the performance-optimized implementation of our binary multiplier template architecture that is speci cally tailored for each combination of FPGA and LEDAcrypt con guration.

75817

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials



**FIGURE 7.** Resource utilization of the proposed multiplier implemented on the Xilinx Artix-7 12 and Xilinx Artix-7 200 FPGAs. Look-Up Table (LUT),Flip-Flop (FF) and Block-RAM (BRAM) resource types are considered. For each resource type, its utilization is expressed as the percentage of the available resources on the target FPGA.

To perform a multiplication, the FPGACtrl module drives the cmdM2S and the weM2S signals to collect the input polynomials from the UART interface. We implemented a blocking communication protocol between the FPGA con-troller and the UART, thus the FPGA controller waits until the UART has sent the required data before closing the com-munication.

Once the input polynomials have been collected, the cmd2Mul signal is used to load the operands into the mul-tiplier and to start the multiplication. For each clock cycle, *BWext* bits of each input polynomial are fed to the multiplierthrough the A and B signals.

The multiplier signals the end of the multiplication through the mulDone signal, while, for each clock cycle, *BWext* bits of the multiplication result are fed to the FPGA controller through the R signal. The cmd2Mul and the ack2Host signals are used to implement the acknowledged protocol to feed the inputs and retrieve the output from the multiplier. Last, the FPGA controller sends the multiplication result to the host computer through the UART by means of the dataM2S data signal. We note that the cmdM2S and the ack2Host signals are used to implement the acknowledged protocol to to exchange the multiplication operands and result between the FPGA controller and the UART module.

**C. AREA AND PERFORMANCE RESULTS**

This section discusses the area and the performance of the proposed multiplier, to demonstrate the scalability and the exibility of our solution over the software implementations.

1) AREA RESULTS

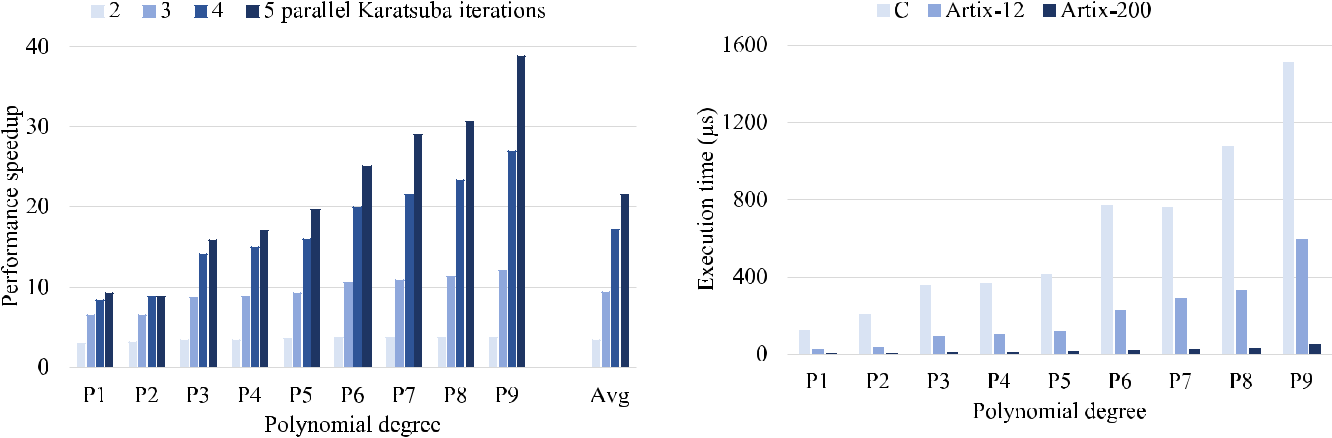
The proposed multiplier exploits a massive BRAM utilization to store partial products and the nal result with two positive side-effects. First, the multiplier can be implemented on tiny FPGAs even for the multiplication of large operands. In particular, the maximum allowed dimension of the operand

in bits is not function of the available amount of ip- ops, that easily become the scarcest resources on small FPGAs, but it is function of the available BRAM storage capacity. We note that a single BRAM can store up to 36kbit and that the smallest considered FPGA features 20 BRAMs. Second, the use of a nested structure to implement the multiplier where storage elements surround the compute stage, optimizes the critical path by construction. In particular, the critical path, which remains independent from the number of implemented Karatsuba iterations, depends on the width (BW) of the com-binational multiplier. This, in turn, determines the critical path of the proposed multiplier, that, however, cannot be improved by a reduction of the value of BW. In fact, any reduction of the value of BW aiming to optimize the critical path of the combinational multiplier generates a much more severe overall performance degradation due to the underuti-lization of the BRAM data-transfer bandwidth.

Figure [7](#page10) reports the normalized resource utilization for each polynomial size of the LEDAcrypt cryptosystem con-sidering the Xilinx Artix-7 12 and the Xilinx Artix-7 200 FPGAs. In particular, for each polynomial size, the percent-age utilization of LUT, ip- ops and BRAM elements is reported. We note that the massive use of BRAM resources minimizes the use of ip- ops, which are therefore never the scarcest resource, while LUT and BRAM utilization are almost aligned even if the reported utilization greatly differs between the two considered FPGAs. For each of the 9 LEDAcrypt con gurations, both the Artix-7 12 and 200 FPGAs have the internal bandwidth of the multiplier set to 64 bits, which corresponds to the bandwidth of the BRAM memories available on the Artix-7 family, making it an optimal choice. For Xilinx Artix-7 12, all the 9 considered polynomials have their optimal hardware con guration with 1 Karatsuba recursion and 3 Comba multipliers, i.e., 3 partial products are computed in parallel. All con gurations almost saturate the FPGA resources in terms of LUT and BRAM, while the low FF utilization, i.e., 8% on average, is due to the

75818 VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials



**FIGURE 8.** Performance speedup with respect to the hardwaremultiplication performed with 1 parallel Karatsuba iteration. Results are shown for hardware multipliers with a number of parallel Karatsuba iterations varying between 2 and 5.

**FIGURE 9.** Execution time (in microseconds) of a multiplication. Resultsare shown for software multiplication on the Intel i7 core and hardware multiplication on the Artix-7 12 and Artix-7 200 FPGAs.

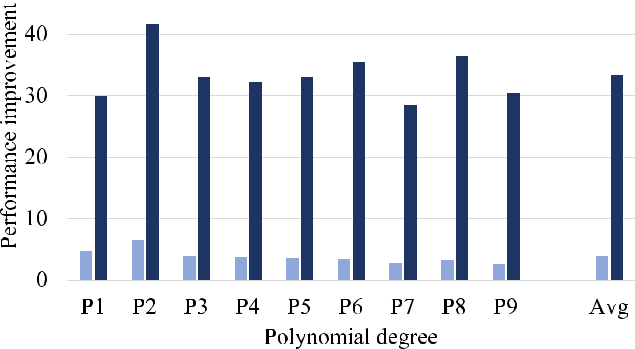


massive use of BRAM to store intermediate values. We note that the unused ip- ops cannot be ef ciently employed, since even all together they cannot contain the information stored in a single BRAM.

For Xilinx Artix-7 200, all the 9 considered polynomials have their optimal hardware con guration with 3 Karatsuba recursion and 27 Comba multipliers. However, the resource utilization for both the LUT and the BRAM is limited to 50%. To better understand this supposedly low resource utilization, we need to analyze the Karatsuba algorithm. In particular, such algorithm allows to substitute a partial product com-putation with a few binary additions in Z2[*x*]. Considering the proposed architecture, a new set of BRAM is used at each iteration of the Karatsuba and additional LUTs are used to perform the additional operations and to compose the intermediate results into the nal partial product. Moreover, the use of too many nested Karatsuba iterations can nega-tively affect the performance since the time spent to split the operands becomes bigger than the time spent to actually perform the Comba multiplication, i.e., Comba operands are too small. To this extent, only the use of a parallel Karatsuba iteration offers a signi cant performance speedup with, how-ever, a non-negligible cost in terms of resource utilization. For each LEDAcrypt con guration, the performance speedup due to the nesting of a parallel Karatsuba iteration is reported in Figure [8.](#page11) The performance speedup is de ned as the ratio between the execution times on a hardware multiplier with only 1 parallel Karatsuba iteration and on hardware multipli-ers with a number of parallel Karatsuba iterations comprised between 2 and 5. We note that its value is always signi cantly positive, while the number of required BRAMs and LUTs grows 3x for each parallel Karatsuba iteration. To this extent, the resource utilization on the Xilinx Artix-7 200 is motivated by the impossibility to add another Karatsuba iteration due to resource limitation, while by using a larger FPGA, such as those of the Xilinx Virtex-7 family, the proposed multiplier can further improve its offered performance.

VOLUME 8, 2020

**FIGURE 10.** Performance improvement with respect to the softwaremultiplication executed on the Intel i7 processor. Results are shown for hardware multiplication on the Artix-7 12 and Artix-7 200 FPGAs.



2) PERFORMANCE RESULTS

Figure [9](#page11) reports the performance, i.e., the execution time, for all the 9 considered polynomial degrees, thus covering all the LEDAcrypt cryptoscheme con gurations. For each polynomial degree, results are reported for the hardware implementations targeting the Artix-7 12 and 200 FPGAs and for the software reference. Considering the LEDAcrypt PKC use case, a multiplication executed with the *gf2x* library takes between 124 and 1510 microseconds, while our hardware multipliers implemented on the Artix-7 12 and 200 FPGAs take respectively between 27 and 597 and between 4 and 50 microseconds. We de ne the performance improvement metric as the ratio between the execution times of a single multiplication on the software reference implementation and on our hardware multipliers.

As shown in Figure [10,](#page11) the Artix-7 200 implementation of the proposed multiplier offers a performance speedup between 28.3 and 41.5 times (33.3 times faster on aver-age) compared to the software implementation. Similarly, the Artix-7 12 implementation of the proposed multiplier offers a performance speedup between 2.5 and 6.4 times (3.6 times faster on average) compared to the software implementation.

75819

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

It is worth noticing that, despite the ad-hoc hardware microar-chitecture, the FPGA implementation works at 143 MHz while the software multiplication executed on an Intel i7 processor clocked at 3.5 GHz.

**V. CONCLUSION**

This work presented a exible and scalable template architec-ture for the hardware implementation of large binary poly-nomial multipliers. The architecture combines the iterative application of the Karatsuba algorithm, to minimize the num-ber of required partial products, with the Comba algorithm, employed to optimize the schedule of their computations.

The design exibility is offered by means of three design-time parameters, that are used to optimally trade performance and resource utilization on a wide range of FPGAs. First, the con gurable number of nested Karatsuba iterations allows to control the reduction of the number of partial products to be computed. Second, the architecture allows to con gure the number of partial products that are computed in parallel. Last, the internal bandwidth of the multiplier architecture is a design parameter as in state-of-the-art digit-serial multipliers.

Moreover, the operand-size scalability is guaranteed in two ways. First, the proposed solution leverages the FPGA block RAMs (BRAMs) to store large operands, the result and the intermediate values. Second, the use of different data widths for the internal multiplier datapath and the primary input-output interface allows to select the optimal performance-resource trade-off point regardless of the size of the input operands.

To demonstrate the effectiveness of our solution, we employed the nine con gurations of the LEDAcrypt public-key cryptosystem as representative use cases for large-degree binary polynomial multiplications. Compared to an opti-mized software implementation employing the *gf2x* C library, our solution demonstrated an average performance speedup of 3.6x and 33.3x, by using the smallest (Artix-7 12) and the largest (Artix-7 200) FPGAs of the Xilinx Artix-7 family, respectively. Moreover, a complete design space exploration demonstrates the possibility of implementing a multiplier for each con guration of the LEDAcrypt on the entire Xilinx Artix-7 family of mid-range FPGAs.

**REFERENCES**

1. N. Sendrier, ‘‘Code-based cryptography: State of the art and perspectives,’’ *IEEE Security Privacy*, vol. 15, no. 4, pp. 44 50, Aug. 2017.
2. E. Berlekamp, R. McEliece, and H. van Tilborg, ‘‘On the inherent intractability of certain coding problems (Corresp.),’’ *IEEE Trans. Inf.* *Theory*, vol. IT-24, no. 3, pp. 384 386, May 1978.
3. R. J. McEliece, ‘‘A public-key cryptosystem based on algebraic coding theory,’’ *DSN Prog. Rep.*, vol. 424, pp. 114 116, Jan./Feb. 1978.
4. M. Baldi, M. Bodrato, and F. Chiaraluce, ‘‘A new analysis of the McEliece cryptosystem based on QC-LDPC codes,’’ in *Proc. Int. Conf. Secur. Cryp-togr. Netw.*, Amal , Italy, 2008, pp. 246 262.
5. R. Misoczki, J.-P. Tillich, N. Sendrier, and P. S. L. M. Barreto, ‘‘MDPC-McEliece: New McEliece variants from moderate density parity-check codes,’’ in *Proc. IEEE Int. Symp. Inf. Theory*, Istanbul, Turkey, Jul. 2013, pp. 2069 2073.
6. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini. *LEDAcrypt Website*. Accessed: Mar. 27, 2020. [Online]. Available:https://www.ledacrypt.org/

75820

1. N. Aragon, P. S. L. M. Barreto, S. Bettaieb, L. Bidoux, O. Blazy, J.-C. Deneuville, P. Gaborit, S. Gueron, T. Güneysu, C. A. Melchor, R. Misoczki, E. Persichetti, N. Sendrier, J.-P. Tillich, V. Vasseur, and G. ZØmor. *BIKE Website*. Accessed: Mar. 27, 2020. [Online]. Available: https://www.bikesuite.org/
2. 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.
3. A. Rezai and P. Keshavarzi, ‘‘High-throughput modular multiplication and exponentiation algorithms using multibit-scan multibit-shift tech-nique,’’ *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 23, no. 9, pp. 1710 1719, Sep. 2015.
4. A. Rezai and P. Keshavarzi, ‘‘High-performance scalable architecture for modular multiplication using a new digit-serial computation,’’ *Microelec-tron. J.*, vol. 55, pp. 169 178, Sep. 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269216303123
5. H. Fan and M. A. Hasan, ‘‘A survey of some recent bit-parallel GF(2*n*) multipliers,’’ *Finite Fields Their Appl.*, vol. 32, pp. 5 43, Mar. 2015.
6. P. G. Comba, ‘‘Exponentiation cryptosystems on the IBM PC,’’ *IBM Syst.* *J.*, vol. 29, no. 4, pp. 526 538, 1990.
7. A. A. Karatsuba and Y. P. Ofman, ‘‘Multiplication of many-digital numbers by automatic computers,’’ *Proc. USSR Acad. Sci. USA*, vol. 145, no. 2, pp. 293 294, 1962.
8. R. P. Brent, P. Gaudry, E. ThomØ, and P. Zimmermann, ‘‘Faster multipli-cation in GF(2)[x],’’ in *Algorithmic Number Theory*, A. J. van der Poorten and A. Stein, Eds. Berlin, Germany: Springer, 2008, pp. 153 166.
9. V. Shoup. *NTL: A Library For Doing Number Theory*.

Accessed: Mar. 27, 2020. [Online]. Available: https://www.shoup.net/ntl/

1. P. Gaudry, L. Sanselme, and E. ThomØ. *MPFQ A Finite Field Library*. Accessed: Mar. 27, 2020. [Online]. Available: http://mpfq.gforge.inria.fr
2. S. Gueron and M. Kounavis, ‘‘Intel carry-less multiplication instruction and its usage for computing the GCM, Revision 2.0,’’ Intel, Santa Clara, CA, USA, Tech. Rep. 323640-001, May 2010. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/carry-less-multiplication-instruction-in-gcm-mode-paper.pdf
3. ‘‘Arm architecture reference manual,’’ Arm, Cambridge, U.K., Tech. Rep., 2020. [Online]. Available: https://developer.arm.com/docs/ddi0487/fa
4. N. Drucker, S. Gueron, and V. Krasnov, ‘‘Fast multiplication of binary polynomials with the forthcoming vectorized VPCLMULQDQ instruc-tion,’’ in *Proc. IEEE 25th Symp. Comput. Arithmetic (ARITH)*, Jun. 2018, pp. 115 119.
5. J. Grossschadl, ‘‘A low-power bit-serial multiplier for nite elds GF(2*m*),’’ in *Proc. ISCAS-IEEE Int. Symp. Circuits Syst.*, vol. 4, May 2001, pp. 37 40.
6. A. Cilardo, ‘‘Fast parallel GF(2*m*) polynomial multiplication for all degrees,’’ *IEEE Trans. Comput.*, vol. 62, no. 5, pp. 929 943, May 2013.
7. B. Rashidi, ‘‘Throughput/area ef cient implementation of scalable polyno-mial basis multiplication,’’ *J. Hardw. Syst. Secur.*, early access, pp. 1 16, Jan. 2020, doi: [10.1007/s41635-019-00087-5.](http://dx.doi.org/10.1007/s41635-019-00087-5)
8. B. Rashidi, S. M. Sayedi, and R. Rezaeian Farashahi, ‘‘Ef cient and low-complexity hardware architecture of Gaussian normal basis multiplication over GF(2*m*) for elliptic curve cryptosystems,’’ *IET Circuits, Devices Syst.*, vol. 11, no. 2, pp. 103 112, Mar. 2017.
9. R. Azarderakhsh and A. Reyhani-Masoleh, ‘‘Low-complexity multiplier architectures for single and hybrid-double multiplications in Gaussian normal bases,’’ *IEEE Trans. Comput.*, vol. 62, no. 4, pp. 744 757, Apr. 2013.
10. J. von Zur Gathen and J. Shokrollahi, ‘‘Ef cient FPGA-based Karatsuba multipliers for polynomials over F2,’’ in *Proc. Int. Workshop Sel. Areas* *Cryptogr.*, Kingston, ON, Canada, Aug. 2005, pp. 359 369, doi:[10.1007/](http://dx.doi.org/10.1007/11693383_25)[11693383\_25.](http://dx.doi.org/10.1007/11693383_25)
11. M. G. Find and R. Peralta, ‘‘Better circuits for binary polynomial multiplication,’’ *IEEE Trans. Comput.*, vol. 68, no. 4, pp. 624 630, Apr. 2019.
12. A. Barenghi, W. Fornaciari, A. Galimberti, G. Pelosi, and D. Zoni, ‘‘Eval-uating the trade-offs in the hardware design of the LEDAcrypt encryption functions,’’ in *Proc. 26th IEEE Int. Conf. Electron., Circuits Syst. (ICECS)*, Nov. 2019, pp. 739 742.
13. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, ‘‘LEDAcrypt: QC-LDPC code-based cryptosystems with bounded decryp-tion failure rate,’’ in *Code-Based Cryptography*, M. Baldi, E. Persichetti, and P. Santini, Eds. Cham, Switzerland: Springer, 2019, pp. 11 43.

VOLUME 8, 2020

D. Zoni *et al.*: Flexible and Scalable FPGA-Oriented Design of Multipliers for Large Binary Polynomials

1. M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, ‘‘Design of LEDAkem and LEDApkc instances with tight parameters and bounded decryption failure rate,’’ Università Politecnica delle Marche, Ancona, Italy, Politecnico di Milano, Milano, Italy, Tech. Rep. [Online]. Available: https://www.ledacrypt.org/archives/of cial\_comment.pdf
2. R. P. Brent, P. Gaudry, E. ThomØ, and P. Zimmermann. *Gf2x Website*. Accessed: Mar. 27, 2020. [Online]. Available: http://gf2x.gforge.inria.fr/

DAVIDE ZONI received the M.Sc. and Ph.D. degrees. He is a Postdoctoral Researcher with the Politecnico di Milano, Italy. He published more than 40 articles in journals and confer-ence proceedings. He received two HiPEAC col-laboration grants, in 2013 and 2014, and two HiPEAC industrial grants, in 2015 and 2017. He is also the Principal Investigator of the Lightweight Application-Speci c Modular Processor (LAMP) Platform (www.lamp-platform.org), that is, a mod-



ular open-hardware RISC-V-based system-on-chip (SoC) speci cally tai-lored to energy-performance trade-off and side-channel security analysis for IoT and low-power embedded devices. His research interests include RTL design and optimization for multicores with particular emphasis on low-power methodologies and hardware-level countermeasures to side-channel attacks.

VOLUME 8, 2020

ANDREA GALIMBERTI received the M.Sc. degree in computer science and engineering from the Politecnico di Milano, Italy, where he is currently pursuing the Ph.D. degree. His research interests include computer architectures, hardware-level countermeasures to side-channel attacks, and the design of hardware accelerators.



WILLIAM FORNACIARI (Senior Member, IEEE) received the M.Sc. and Ph.D. degrees. He is an Associate Professor with the Politecnico di Milano, Italy. He published six books and around 300 articles in international journals and confer-ences, collecting six best paper awards and one certi cation of appreciation from the IEEE. He has been the coordinator of both FP7 and H2020 EU Projects. He holds three international patents on low-power design. His research interests include



embedded and cyber-physical systems, the energy-aware design of software and hardware, the run-time management of resources, high-performance computing, design optimization, and the thermal management of multi-many cores. In 2016, he won the HiPEAC Technology Transfer Award.



75821