Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

Flexible and scalable FPGA-oriented design of multipliers for large binary polynomials

DAVIDE ZONI[1], ANDREA GALIMBERTI[1], AND WILLIAM FORNACIARI[1], (Senior Member, IEEE)

[1]Dipartimento di Elettronica Informazione e Bioingegneria (DEIB), Politecnico di Milano, 20133 Milano, Italy
(e-mail: {davide.zoni,andrea.galimberti,william.fornaciari}@polimi.it)

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

This work was supported by the EU H2020 "RECIPE" Project (801137)

ABSTRACT With the recent advances in quantum computing, code-based cryptography is foreseen to be one 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 efficient multipliers is crucial to optimize the performance of post-quantum public-key cryptographic solutions. This manuscript presents a flexible 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 configurations of the LEDAcrypt public-key cryptosystem as representative use cases for large-degree binary polynomial multiplications. For each configuration 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 configurations 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 confidentiality 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 resistant 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 attacks performed with quantum computers imposes a change in the underlying computationally hard problem.

Code-based cryptography emerged as one of the few mathematical 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 finding a fixed 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 first cryptosystem relying on the hardness of the decoding problem [3], while several following proposals aimed at finding code families with efficient state-space representation to reduce the size of the cryptographic 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 cyclically rotating the first one. The arithmetic of circulant matrices with size p can be proven to be isomorphic to the arithmetic of the polynomials modulo $x^p + 1$ over the same field as the coefficients of the circulant matrices. In particular, in the case of binary linear block codes, the arithmetic of $p \times p$ circulant matrices over $\mathbb{Z}_2$ can be substituted by the arithmetic of polynomials in $\mathbb{Z}_2[x]/(x^p + 1)$, thus reducing significantly 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 efficient multiplication primitive for the performance of quantum-resistant public-key cryptosystems. In particular, available software solutions for different QC-LDPC codes involved in the NIST standardization process, e.g., LEDAcrypt [6] and BIKE [7], demonstrate the impossibility to cope with the required performance, especially 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 integers 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 target modular integer multiplication [8]–[10], while code-based cryptography leverages the field of binary polynomials. Second, the effectiveness of available binary polynomial multipliers 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 efficiently scale up to support the key-size of code-based cryptoschemes, i.e., dozens of thousands of bits.

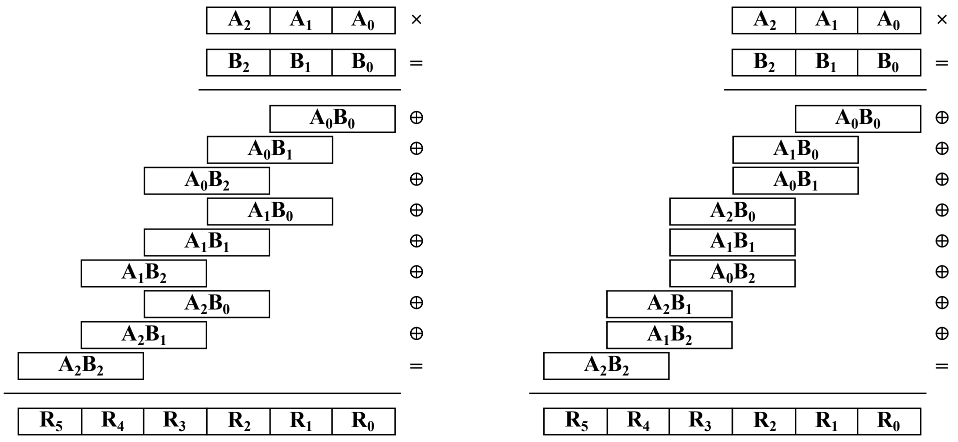
Contributions - The manuscript presents an FPGA-oriented hardware design methodology to implement multipliers for large-degree binary polynomials. The possibility of employing the proposed design in post-quantum code-based cryptosystem implementations motivated its assessment against the Xilinx Artix-7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

- Template architecture flexibility - our template multiplier architecture exposes three parameters to optimally trade, at design-time, performance and resource utilization on a wide range of FPGAs. First, the configurable number of nested Karatsuba iterations allows to control the reduction of the number of partial products to be computed. Second, it is possible to configure 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.
  For each of the polynomial configurations of the LEDAcrypt post-quantum public-key cryptosystem, performance results are collected considering the implementation of the proposed template multiplier on different 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 multiplier 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 flip-flops, to store the operands, the result, and the intermediate values, allows to manage 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 configured to use different bandwidths.
  Our exhaustive design space exploration demonstrates the possibility of implementing a performance-optimized multiplier, for each configuration 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 discusses the background on multiplication algorithms in the finite field of binary polynomials and summarizes the state-of-the-art. The proposed multiplier design is discussed in Section III. Section IV presents the experimental results. Conclusions are drawn in Section V.

II. BACKGROUND AND RELATED WORKS

To support cryptographic computations, the state-of-the-art proposes several designs of efficient multipliers implemented either in software or in hardware. The rest of this section is organized in two parts. Section II-A presents the theoretical background on the finite field 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 presents the state-of-the-art in terms of hardware and software implementations of the multipliers in such finite fields.



(a) 3-digit schoolbook (b) 3-digit Comba

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.

A. BACKGROUND ON BINARY POLYNOMIAL MULTIPLICATION

A finite field, also called Galois field, is a set that contains a finite number of elements on which the addition, subtraction, multiplication and division operations are defined. $\mathbb{Z}_2$, or $GF(2)$, is the Galois field of two elements, i.e., the smallest Galois field. The two elements of $\mathbb{Z}_2$ are usually referred to as 0 and 1, and they are respectively the additive and the multiplicative identities. The field's addition operation corresponds to the logical XOR operation, while the multiplication operation corresponds to the logical AND operation.

Polynomials with coefficients in $\mathbb{Z}_2$, i.e., 0 and 1, form a Galois field, which is commonly referred to as $\mathbb{Z}_2[x]$ or $GF(2)[x]$. The addition of two elements of such field 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 polynomial, which is taken from the construction of the field. For example, $\mathbb{Z}_2[x]/(x^p + 1)$ is the Galois field of polynomials with coefficients in $\mathbb{Z}_2$ for which the irreducible polynomial is $x^p + 1$, thus polynomials which belong to such field have degree at most equal to $p - 1$.

Multiplication in $\mathbb{Z}_2[x]$ conceptually works like long multiplication between integer numbers, except for the fact that the carry is always discarded instead of added to the more significant position. This property derives from the fact that the addition in $\mathbb{Z}_2$ corresponds to the logical XOR. For this reason, the multiplication operation in $\mathbb{Z}_2[x]$ is also commonly referred to as carry-less multiplication.

Considering the quasi-cyclic codes employed in many proposals for post-quantum code-based cryptosystems, the arithmetic of $p \times p$ circulant matrices over $\mathbb{Z}_2$ can be substituted with the arithmetic of polynomials in $\mathbb{Z}_2[x]/(x^p + 1)$. In code-based cryptosystems, matrix multiplication is the most computationally intensive operation of the encryption primitives. Since matrix multiplication corresponds to polynomial multiplication 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 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.

Schoolbook multiplication - The schoolbook multiplication method implements the long multiplication for the execution on a digital system. Starting from the binary representation of the $\mathbb{Z}_2[x]$ 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 polynomial 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.

Comba multiplication - The Comba multiplication

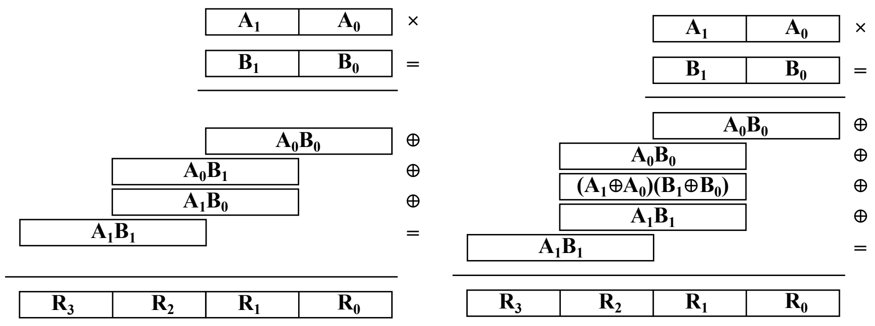


FIGURE 2: Multiplication of two-digit polynomials considering schoolbook and Karatsuba algorithms. Karatsuba optimizes the computation by leveraging the intuition for which multiplications are more computationally expensive than additions in $\mathbb{Z}_2[x]$. In particular, schoolbook requires four multiplications and three additions, while Karatsuba performs the same computation using three multiplications and six additions.

method [12] minimizes the number of memory writes required by the schoolbook method by optimizing the order of computation of the partial products (see Figure 1b). We note that the Comba algorithm requires exactly the same number of partial products and corresponding 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. For example, the final value for the R_0 digit is written in memory when the A_0B_0 partial product is ready. Moreover, the Comba method operates a right shift of digit size to trash the lower part of the A_0B_0 partial product, since it is not necessary anymore. In a similar manner, the final value for the R_1 digit is written in memory when the subsequent A_0B_1 and A_1B_0 have been computed. Again, the lower part of the intermediate result is trashed out since it is no longer useful. To this extent, the Comba method ensures a maximum of $2 \times size(digit)$ bits for the intermediate result, while the schoolbook algorithm requires at least $N \times size(digit)$ 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.

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 computationally expensive than the addition in $\mathbb{Z}_2[x]$. Figure 2 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.

B. RELATED WORKS ON BINARY POLYNOMIAL MULTIPLIERS

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

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 $\mathbb{Z}_2$, implementing several multiplication algorithms to optimize the computation 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 specifically tailored to deliver top-notch performance for finite-fields 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 architecture, 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 instruction on ARM targets [18]. The work in [19] leverages the VPCLMULQDQ instruction that will be supported in future Intel Ice Lake solutions, and which is the vectorized extension 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 2^{16} predict a 2x speed-up compared to current computing platforms.

Hardware accelerators - The state-of-the-art contains several architectures implementing ad-hoc hardware accelerators for the Galois field 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 particular, such hardware accelerators output the M-bit result after M clock cycles, thus their latency strictly depends on the size of the input. [20] presents a low-power bit-serial multiplier architecture for binary polynomials for which an M-bit multiplier implementation requires $28 \times M$ gates. The limited performance and flexibility 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. [21] details the realization of the optimal bit-parallel design given the structure of the target binary polynomial Galois field, i.e., the size of the polynomials of the field and its associated irreducible polynomial.

We note that all bit-parallel solutions leverage the size of the operands to deliver efficient ad-hoc architectures. To this end, each architecture is customized for a specific Galois field and it is therefore not reusable. The limited flexibility 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 polynomial basis multipliers offer a superior design flexibility. In particular, the operands are organized in digits, i.e., chunks with a fixed number of bits, and the multiplication proceeds on a digit-by-digit basis. The possibility to configure the size of the digit allows to trade the performance with the resource utilization. [22] presents a low-area and scalable digit-serial architecture to perform polynomial basis multiplications over $\mathbb{Z}_2[x]$. Two digit-serial architectures for multiplication over Galois fields 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 complexity 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 solutions is limited to the possibility of configuring 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 flexibility of current digit-serial architectures with the use of fast multiplication algorithms to aggressively reduce the number of computed 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 performance. [25] proposes a hardware multiplier employing an ad-hoc implementation of the Karatsuba algorithm 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 fixed 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. [27] compares two implementations of binary polynomial multipliers targeting the encryption function of LEDAcrypt. Depending on the actual number of coefficients set to 1 in one of the two polynomials, 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 configurable number of times, and the architecture 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 flexible 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 specific FPGA. Moreover, its peculiar flexibility 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 polynomials even on resource constrained FPGA targets. Figure 3 depicts the architectural top view of the proposed multiplier (`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 configurable bandwidth, BW_{ext}, as well as input and output memory

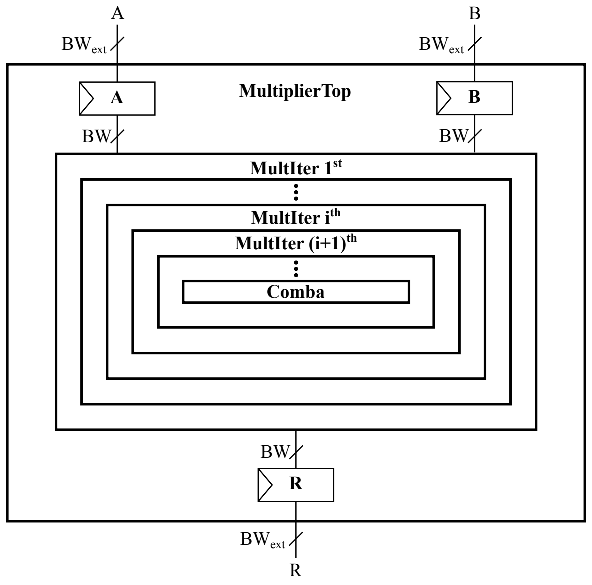


FIGURE 3: Top view of the proposed multiplier template.

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 bandwidth (BW_{ext}). In particular, the former has no externally 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 operate on large polynomials, since no physical interface can accommodate a datapath width of dozens of thousands of bits.

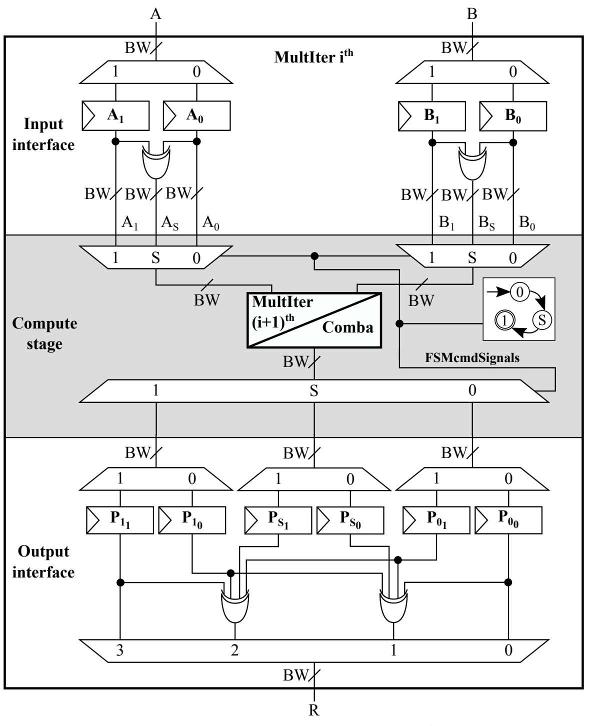
The architecture of the `MultiplierTop` module allows to implement a configurable number of iterations of the Karatsuba algorithm, aggressively reducing the number of required partial products (see the `MultIter` blocks in Figure 3). At the end of the recursive application of the Karatsuba algorithm, the Comba multiplication algorithm performs the actual computation of the partial products (see `Comba` in Figure 3). 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 recursive application of the Karatsuba iterations is still too large to fit into the combinational $BW \times BW$ 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 details the architecture that allows to recursively apply the Karatsuba algorithm for a predefined number of times. Such structure is meant to minimize the number of required partial products and to maximize the level of parallelism to compute the remaining partial products. Section III-B discusses the architecture to actually compute the partial products. Depending on the required performance-resources trade-off, such configurable computing architecture 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.

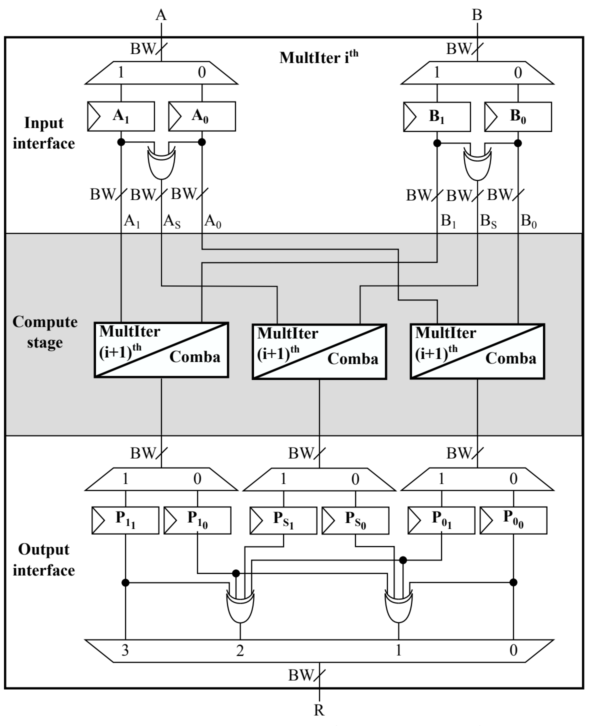
A. KARATSUBA MULTIPLIER ARCHITECTURE

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 algorithms.

Figure 4 depicts the architecture of two nested Karatsuba iterations, i^{th} and $(i+1)^{th}$, which is at the core of the iterative 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 flexibility 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 multiplication. The input interface splits each one of the two polynomials 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 final 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 configuration 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 compute 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 application 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 parallel or serial implementation of the compute stage represents a configuration parameter of the proposed template multiplier. The parallel implementation of the compute stage only requires a proper connection of the input and the output signals to the nested `MultIter/Comba` modules (see Figure 4b). The serial implementation of the compute



(a) Serial architecture of nested i^{th} and $(i+1)^{th}$ Karatsuba iterations



(b) Parallel architecture of nested i^{th} and $(i+1)^{th}$ Karatsuba iterations

FIGURE 4: The proposed architecture implements a configurable number of nested Karatsuba algorithm iterations. Each iteration can be implemented 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. A_i, B_i and P_{j_k} are memories with a BW-bit bandwidth.

stage must orchestrate the computation of the three partial products by leveraging the single, i.e., shared, computing block (`MultIter/Comba`) (see Figure 4a). To this purpose, a simple finite-state-machine drives the multiplexing infrastructures to forward the correct part of the operands from the storage elements of the i^{th} `MultIter` module to the single compute unit, i.e., $(i+1)^{th}$ `MultIter/Comba`.
In summary, the proposed template multiplier architecture allows to flexibly configure i) the number of Karatsuba iterations 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 depicts the architecture of the `Comba` module,

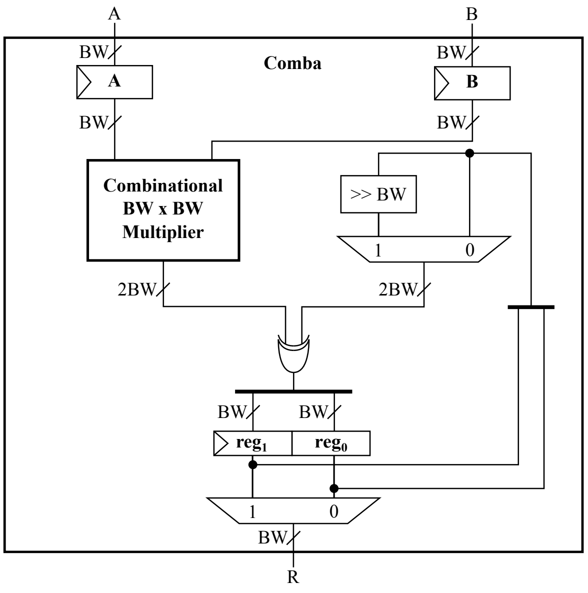


FIGURE 5: The architecture of the proposed Comba multiplier. A and B are memories with a BW-bit bandwidth, reg_i are BW-bit registers.

Algorithm 1 Bit-level combinational multiplication. A and B are BW-bit digits, R is $(2BW-1)$-bit long. $A[i]$, $B[i]$ and $R[i]$ indicate single bits.

```
1: function $[R]$ CombinationalMul($A, B$)
2:     for $i \in 0 : BW-1$ do
3:         for $j \in 0 : BW-1$ do
4:             $R[i+j] = R[i+j] \oplus (A[i] \cdot B[j])$;
5:         end for
6:     end for
7: end function
```

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 algorithm 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, thus far too large to perform a single combinational multiplication. 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` module, the `Combinational BW x BW Multiplier` performs the multiplication between two digits (see Figure 5). In particular, Algorithm 1 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 $BW \times BW$ multiplications 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 significant bits of the XOR of partial products computed by the combinational multiplier. When the computation of the sum is completed, the least significant BW bits, i.e., the BW bits stored in reg_0, are committed to the output of the Comba Multiplier, while the most significant 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 (execution time) with the final goal of highlighting the flexibility and the scalability of our solution. To demonstrate the flexibility, 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. Performance 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 overviews the LEDAcrypt cryptosystem, with emphasis on both the operand size and the computational impact of the polynomial multiplication. Section IV-B details the experimental settings encompassing both hardware and software. Last, the experimental results in terms of resource utilization and performance are discussed in Section IV-C.

A. LEDACRYPT CRYPTOSYSTEM

We consider the Public-Key Cryptosystem (PKC) from the LEDAcrypt post-quantum cryptography suite [28] as a representative use case for a large-degree binary polynomial multiplier. 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 finalists in the National Institute of Standards and Technology (NIST) initiative 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 $n_0 \in \{2,3,4\}$ 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 configurations corresponds a polynomial degree p (see Table 1).

The encryption function of the LEDAcrypt PKC takes as its inputs a plaintext message composed as a $1 \times p(n_0-1)$ binary vector u, an error vector composed as a $1 \times pn_0$ binary vector e and a public key G structured as $(n_0-1) \times n_0$ circulant blocks with size $p \times p$.

The ciphertext c is then computed as:

$$c = u \cdot G + e$$

Because of the systematic form and the quasi-cyclic nature of G, the $u \cdot G$ matrix multiplication corresponds to $n_0 - 1$ polynomial multiplications in $\mathbb{Z}_2[x]/(x^p+1)$. These $n_0 - 1$ 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 configurations of the LEDAcrypt PKC, i.e., binary polynomial multiplications have been evaluated for polynomials of degree p for each configuration [29]. LEDAcrypt configurations and their corresponding polynomial degrees are shown in more detail in Table 1.

TABLE 1: Polynomial degrees of LEDAcrypt PKC configurations.

Security Level	n_0	Label	Degree
AES-128	2	P4	15013
	3	P2	9643
	4	P1	8467
AES-192	2	P7	24533
	3	P5	17827
	4	P3	14717
AES-256	2	P9	37619
	3	P8	28477
	4	P6	22853

B. EXPERIMENTAL SETUP

Hardware setup - The architecture for binary polynomial multiplication discussed in Section III 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). Each design has been implemented considering 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 configuration, i.e., the feasible one providing the best performance in terms of the time to compute the entire multiplication. Such configurations have been identified 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 fixed at 64 bits, to match the bandwidth of the BRAM memories implemented in Xilinx Artix-7 FPGAs, thus delivering the optimal data transfer.

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

	Artix-7 12	Artix-7 200
LUT	8000	134600
FF	16000	269200
BRAM	20	365

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 multiplications. To provide a representative comparison, we tuned the library according trough its automatic tuning procedure to select the most appropriate multiplication algorithm depending on the underlying hardware and polynomial degree. The experimental evaluation of the software-implemented multiplication has been carried out on an Intel Core i7-6700HQ processor forcing a fixed operating frequency of 3.5 GHz to avoid performance variability due to the power management controller. For each value of p as defined 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.

Functional validation - The functional validation is meant to check the correctness of the multiplication results obtained from the hardware implementation of our multiplier template architecture. We employed the *gf2x* C library [30] (version 1.3.0) as the reference for our functional validation. In particular, for each polynomial degree defined in the LEDAcrypt public-key cryptosystem, i.e. the 9 configurations reported in Table 1, we collected the results of the software execution of 10000 multiplications 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 implementations 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 multiplier 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 configurations and the three considered FPGAs, i.e., Xilinx Artix-7 12, Xilinx Artix-7 100 and Xilinx 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 unified validation infrastructure for both the post-implementation (timing) simulations and the board prototype executions (see Figure 6). The `FVA` is made of three parts. The FPGA controller (`FPGACtrl`) communicates with the host computer to collect the input polynomials and to return the multiplication results. The UART module (`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 specifically tailored for each combination of FPGA and LEDAcrypt configuration.

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 controller and the UART, thus the FPGA controller waits until the UART has sent the required data before closing the communication.

Once the input polynomials have been collected, the

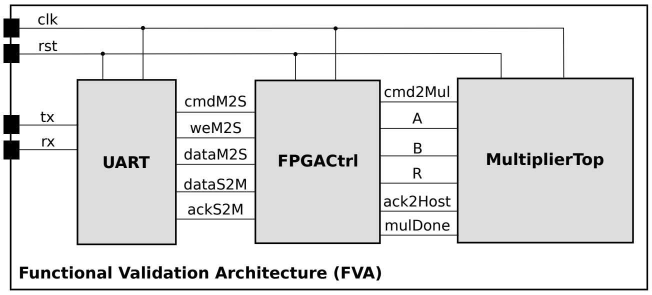


FIGURE 6: Hardware setup for the functional assessment of the proposed binary multiplier . The hardware setup is made of three parts. The UART module allows the communication between the host computer and the `Functional Validation Architecture`. The FPGA controller (`FPGACtrl`) coordinates the communication 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.

`cmd2Mul` signal is used to load the operands into the multiplier and to start the multiplication. For each clock cycle, BW_{ext} bits of each input polynomial are fed to the multiplier through the `A` and `B` signals.
The multiplier signals the end of the multiplication through the `mulDone` signal, while, for each clock cycle, BW_{ext} 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.

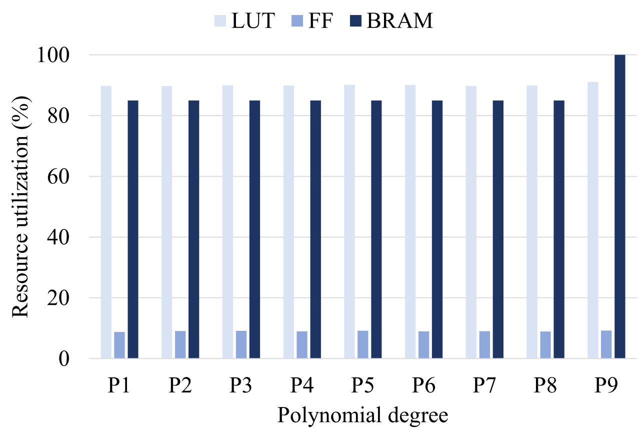
C. AREA AND PERFORMANCE RESULTS

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

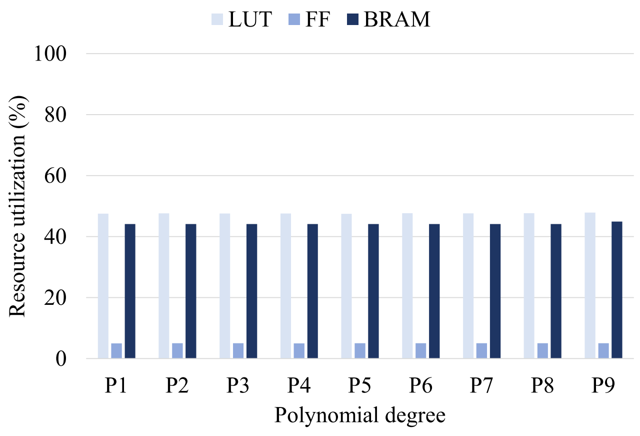
Area results - The proposed multiplier exploits a massive BRAM utilization to store partial products and the final result with two positive side-effects. First, the multiplier can be implemented on tiny FPGAs even for the multiplication of large operands. In particular, the maximum allowed dimension of the operand in bits is not function of the available amount of flip-flops, 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 combinational 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 underutilization of the BRAM data-transfer bandwidth.

Figure 7 reports the normalized resource utilization for each polynomial size of the LEDAcrypt cryptosystem considering the Xilinx Artix-7 12 and the Xilinx Artix-7 200 FPGAs. In particular, for each polynomial size, the percentage utilization of LUT, flip-flops and BRAM elements is reported. We note that the massive use of BRAM resources minimizes the use of flip-flops, 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 configurations, 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 configuration with 1 Karatsuba recursion and 3 Comba multipliers, i.e., 3 partial products are computed in parallel. All configurations 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 massive use of BRAM to store intermediate values. We note that the unused flip-flops cannot be efficiently 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 configuration 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 computation with a few binary additions in $\mathbb{Z}_2[x]$. 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 final partial product. Moreover, the use of too many nested Karatsuba iterations can negatively 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 significant performance speedup with, however, a non-negligible cost in terms of resource utilization. For each LEDAcrypt configuration, the performance speedup due to the nesting of a parallel Karatsuba iteration is reported in Figure 8. The performance speedup is defined as the ratio



(a) Xilinx Artix-7 12



(b) Xilinx Artix-7 200

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.

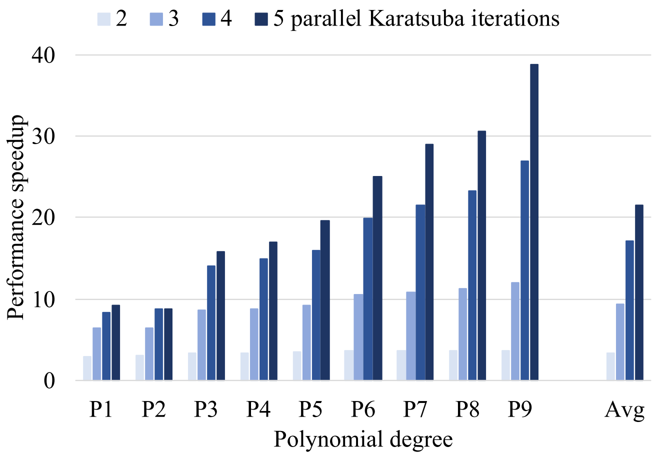


FIGURE 8: Performance speedup with respect to the hardware multiplication performed with 1 parallel Karatsuba iteration. Results are shown for hardware multipliers with a number of parallel Karatsuba iterations varying between 2 and 5.

between the execution times on a hardware multiplier with only 1 parallel Karatsuba iteration and on hardware multipliers with a number of parallel Karatsuba iterations comprised between 2 and 5. We note that its value is always significantly 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.

Performance results - Figure 9 reports the performance, i.e., the execution time, for all the 9 considered polynomial degrees, thus covering all the LEDAcrypt cryptoscheme configurations. 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 define the

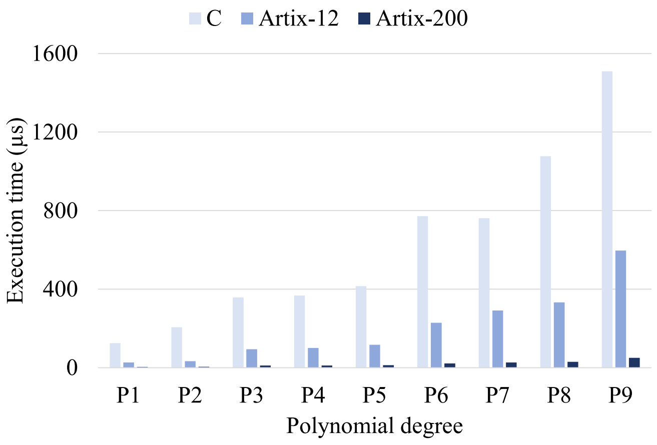


FIGURE 9: Execution time (in microseconds) of a multiplication. Results are shown for software multiplication on the Intel i7 core and hardware multiplication on the Artix-7 12 and Artix-7 200 FPGAs.

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

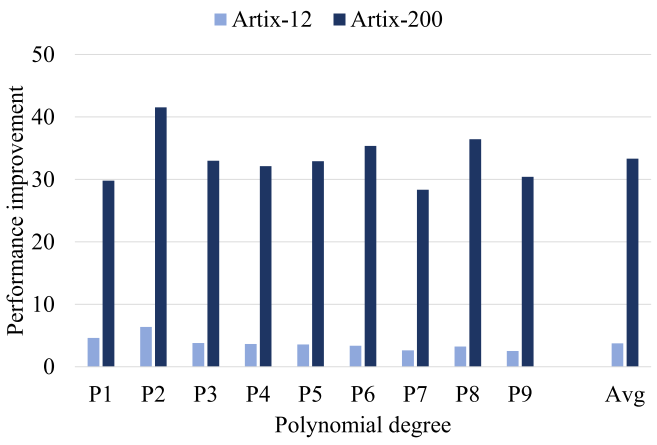


FIGURE 10: Performance improvement with respect to the software multiplication executed on the Intel i7 processor. Results are shown for hardware multiplication on the Artix-7 12 and Artix-7 200 FPGAs.

V. CONCLUSIONS

This work presented a flexible and scalable 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, employed to optimize the schedule of their computations.

The design flexibility 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 configurable number of nested Karatsuba iterations allows to control the reduction of the number of partial products to be computed. Second, the architecture allows to configure 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 configurations of the LEDAcrypt public-key cryptosystem as representative use cases for large-degree binary polynomial multiplications. Compared to an optimized 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 configuration 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, 2017.
[2] E. Berlekamp, R. McEliece, and H. van Tilborg, “On the inherent intractability of certain coding problems (corresp.),” IEEE Transactions on Information Theory, vol. 24, no. 3, pp. 384–386, May 1978.
[3] R. J. McEliece, “A Public-Key Cryptosystem Based on Algebraic Coding Theory,” DSN Progress Report, pp. 114–116, 1978.
[4] M. Baldi, M. Bodrato, and F. Chiaraluce, “A New Analysis of the McEliece Cryptosystem Based on QC-LDPC Codes,” in Security and Cryptography for Networks, 6th Int.’l Conference, SCN 2008, Amalfi, Italy, Sep. 10-12, 2008. Proc., 2008.
[5] R. Misoczki, J. Tillich, N. Sendrier, and P. S. L. M. Barreto, “MDPC-McEliece: New McEliece variants from Moderate Density Parity-Check codes,” in Proc. of the 2013 IEEE Int.’l Symp. on Inf. Theory, Istanbul, Turkey, July 7-12, 2013, 2013.
[6] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, “LEDAcrypt website,” https://www.ledacrypt.org/.
[7] 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. Zémor, “BIKE website,” https://www.bikesuite.org/.
[8] C. Rafferty, M. O’Neill, and N. Hanley, “Evaluation of large integer multiplication methods on hardware,” IEEE Transactions on Computers, vol. 66, no. 8, pp. 1369–1382, Aug 2017.
[9] A. Rezai and P. Keshavarzi, “High-Throughput Modular Multiplication and Exponentiation Algorithms Using Multibit-Scan-Multibit-Shift Technique,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 9, pp. 1710–1719, 2015.
[10] A. Rezai and P. Keshavarzi, “High-performance scalable architecture for modular multiplication using a new digit-serial computation,” Microelectronics Journal, vol. 55, pp. 169 – 178, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0026269216303123
[11] H. Fan and M. A. Hasan, “A Survey of Some Recent Bit-Parallel GF (2^n) Multipliers,” Finite Fields Appl., vol. 32, no. C, Mar. 2015.
[12] P. G. Comba, “Exponentiation cryptosystems on the ibm pc,” IBM Systems Journal, vol. 29, no. 4, pp. 526–538, 1990.
[13] A. Karatsuba and Y. Ofman, “Multiplication of many-digital numbers by automatic computers,” Proceedings of the USSR Academy of Sciences, vol. 145, pp. 293–294, 1962.
[14] R. P. Brent, P. Gaudry, E. Thomé, and P. Zimmermann, “Faster multiplication in gf(2)[x],” in Algorithmic Number Theory, A. J. van der Poorten and A. Stein, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 153–166.
[15] V. Shoup, “Ntl: A library for doing number theory,” https://www.shoup.net/ntl/.
[16] P. Gaudry, L. Sanselme, and E. Thomé, “Mpfq - a finite field library,” http://mpfq.gforge.inria.fr.
[17] S. Gueron and M. Kounavis, “Intel Carry-Less Multiplication Instruction and its Usage for Computing the GCM,” Intel, Tech. Rep., May 2010, https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/carry-less-multiplication-instruction-in-gcm-mode-paper.pdf.
[18] “Arm architecture reference manual,” Arm, Tech. Rep., 2020, https://developer.arm.com/docs/ddi0487/fa.
[19] N. Drucker, S. Gueron, and V. Krasnov, “Fast multiplication of binary polynomials with the forthcoming vectorized vpclmulqdq instruction,” in 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), June 2018, pp. 115–119.
[20] J. Grossschadl, “A low-power bit-serial multiplier for finite fields gf(2/sup m/),” in ISCAS 2001. The 2001 IEEE International Symposium on Circuits and Systems (Cat. No.01CH37196), vol. 4, May 2001, pp. 37–40 vol. 4.

[21] A. Cilardo, "Fast Parallel GF(2^m) Polynomial Multiplication for All Degrees," IEEE Transactions on Computers, vol. 62, no. 5, pp. 929–943, May 2013.
[22] B. Rashidi, "Throughput/area efficient implementation of scalable polynomial basis multiplication," in Journal of Hardware and Systems Security, Jan 2020.
[23] B. Rashidi, S. M. Sayedi, and R. Rezaeian Farashahi, "Efficient and low-complexity hardware architecture of Gaussian normal basis multiplication over GF(2m) for elliptic curve cryptosystems," IET Circuits, Devices Systems, vol. 11, no. 2, pp. 103–112, 2017.
[24] R. Azarderakhsh and A. Reyhani-Masoleh, "Low-complexity multiplier architectures for single and hybrid-double multiplications in gaussian normal bases," IEEE Transactions on Computers, vol. 62, no. 4, pp. 744–757, April 2013.
[25] J. von zur Gathen and J. Shokrollahi, "Efficient FPGA-Based Karatsuba Multipliers for Polynomials over F_2," in Selected Areas in Cryptography, 12th International Workshop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, 2005, pp. 359–369. [Online]. Available: https://doi.org/10.1007/11693383_25
[26] M. G. Find and R. Peralta, "Better circuits for binary polynomial multiplication," IEEE Transactions on Computers, vol. 68, no. 4, pp. 624–630, April 2019.
[27] A. Barenghi, W. Fornaciari, A. Galimberti, G. Pelosi, and D. Zoni, "Evaluating the trade-offs in the hardware design of the ledacrypt encryption functions," in 2019 26th IEEE International Conference on Electronics, Circuits and Systems (ICECS), Nov 2019, pp. 739–742.
[28] M. Baldi, A. Barenghi, F. Chiaraluce, G. Pelosi, and P. Santini, "LEDAcrypt: QC-LDPC Code-Based Cryptosystems with Bounded Decryption Failure Rate," in Code-Based Cryptography, M. Baldi, E. Persichetti, and P. Santini, Eds. Cham: Springer International Publishing, 2019, pp. 11–43.
[29] ——, "Design of LEDAkem and LEDApkc instances with tight parameters and bounded decryption failure rate," https://www.ledacrypt.org/archives/official_comment.pdf.
[30] R. P. Brent, P. Gaudry, E. Thomé, and P. Zimmermann, "gf2x website," http://gf2x.gforge.inria.fr/.

DAVIDE ZONI , MSc, PhD, is a Post-doc Researcher at Politecnico di Milano, Italy. He published more than 40 papers in journals and conference proceedings. His research interests include RTL design and optimization for multi-cores with particular emphasis on low power methodologies and hardware-level countermeasures to side-channel attacks. He received two HiPEAC collaboration grants in 2013 and 2014 and two HiPEAC industrial grant in 2015 and 2017. He is also the principal investigator of the Lightweight Application-specific Modular Processor (LAMP) platform (www.lamp-platform.org), that is a modular open-hardware RISC-V based System-on-Chip (SoC) specifically tailored to energy-performance trade-off and side-channel security analysis for IoT and low-power embedded devices.

ANDREA GALIMBERTI , MSc, is a PhD student at Politecnico di Milano, Italy. He received his MSc degree in 2019 in Computer Science and Engineering at Politecnico di Milano. His research interests include computer architectures, hardware-level countermeasures to side-channel attacks and design of hardware accelerators.

WILLIAM FORNACIARI , MSc, PhD, IEEE senior member, is Associate Professor at Politecnico di Milano, Italy. He published 6 books and around 300 papers in int. journals and conferences, collecting 6 best paper awards and one certification of appreciation from IEEE. He holds three international patents on low power design. He has been coordinator of both FP7 and H2020 EU-projects. In 2016 he won the HiPEAC Technology Transfer Award. His research interests include embedded and cyber-physical systems, energy-aware design of sw and hw, run-time management of resources, High-Performance-Computing, design optimization and thermal management of multi-many cores.