Hindawi
Journal of Sensors
Volume 2021, Article ID 8860413, 13 pages
https://doi.org/10.1155/2021/8860413

Research Article

A Compact FPGA-Based Accelerator for Curve-Based Cryptography in Wireless Sensor Networks

Miguel Morales-Sandoval,[1] **Luis Armando Rodriguez Flores**,[2] **Rene Cumplido**,[2] **Jose Juan Garcia-Hernandez**,[1] **Claudia Feregrino**,[2] **and Ignacio Algredo**[2]

[1]*Centro de Investigacion y de Estudios Avanzados-Cinvestav Tamaulipas, Mexico*
[2]*Instituto Nacional de Astrofisica, Optica y Electronica-INAOE, Mexico*

Correspondence should be addressed to Miguel Morales-Sandoval; miguel.morales@cinvestav.mx

Received 17 April 2020; Revised 12 September 2020; Accepted 30 November 2020; Published 6 January 2021

Academic Editor: Iftikhar Ahmad

The main topic of this paper is low-cost public key cryptography in wireless sensor nodes. Security in embedded systems, for example, in sensor nodes based on field programmable gate array (FPGA), demands low cost but still efficient solutions. Sensor nodes are key elements in the Internet of Things paradigm, and their security is a crucial requirement for critical applications in sectors such as military, health, and industry. To address these security requirements under the restrictions imposed by the available computing resources of sensor nodes, this paper presents a low-area FPGA-prototyped hardware accelerator for scalar multiplication, the most costly operation in elliptic curve cryptography (ECC). This cryptoengine is provided as an enabler of robust cryptography for security services in the IoT, such as confidentiality and authentication. The compact property in the proposed hardware design is achieved by implementing a novel digit-by-digit computing approach applied at the finite field and curve level algorithms, in addition to hardware reusing, the use of embedded memory blocks in modern FPGAs, and a simpler control logic. Our hardware design targets elliptic curves defined over binary fields generated by trinomials, uses fewer area resources than other FPGA approaches, and is faster than software counterparts. Our ECC hardware accelerator was validated under a hardware/software codesign of the Diffie-Hellman key exchange protocol (ECDH) deployed in the IoT MicroZed FPGA board. For a scalar multiplication in the *sect233* curve, our design requires 1170 FPGA slices and completes the computation in 128820 clock cycles (at 135.31 MHz), with an efficiency of 0.209 kbps/slice. In the codesign, the ECDH protocol is executed in 4.1 ms, 17 times faster than a MIRACL software implementation running on the embedded processor Cortex A9 in the MicroZed. The FPGA-based accelerator for binary ECC presented in this work is the one with the least amount of hardware resources compared to other FPGA designs in the literature.

1. Introduction

Nowadays, the computing paradigm of Internet of Things (IoT) is enabling a large number of applications in wireless technologies such as smart vehicles, smart buildings, health monitoring, energy management, environmental monitoring, food supply chains, and manufacturing [1].

In critical IoT applications, as in the Industrial Internet of Things (IIoT) or in healthcare (Medical Internet of Things—MIoT), embedded system devices have become an integral part [2] and easy targets of attacks, mainly because they are physically more accessible. Cyberphysical systems in these domains create new classes of risks resulting from their interaction between cyberspace and the physical world. Wireless sensor networks (WSN) are the cornerstone for realizations of IoT applications, where in some cases, the data generated, stored, or transmitted by the nodes (i.e., embedded systems) require robust security mechanisms to provide them with security services of confidentiality, authentication, integrity, and nonrepudiation. Consider the model for a set of networked IoT devices (for example, a wireless sensor network) in Figure 1. Security risks arise since a malicious node can get unauthorized access to (sensible) data, maliciously alter data, and impersonate legitimate nodes, thus posing

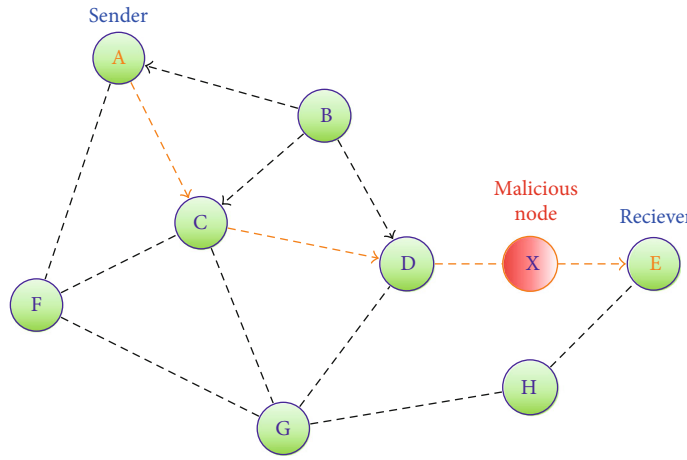

Figure 1: Simplified model of networked IoT devices collecting and sharing data.

threats to confidentiality and authentication in the communication path between a sender and a receiver node.

A robust approach to provide such security services in the IoT domain is the public key cryptography (PKC). PKC in its different families is based on mathematical problems, and underlying realizations involve costly arithmetic algorithms over finite fields, rings, or groups. In the literature, a vast amount of research has focused in hardware acceleration of PKC at the different levels of involved arithmetic algorithms. The main approaches for hardware implementations of PKC have focused on speeding up the underlying group and finite field operations at the expense of a high amount of hardware resources. However, the main drawback with hardware for PKC in WSN is the long key lengths which amount to large chip area, circuit delays, and increased power dissipation [3].

The hardware implementation of PKC-based security solutions in resource-constrained devices typically found in IoT scenarios, as in FPGA-based sensor nodes, and using a straightforward approach is not viable. Lightweight cryptography (LWC) [4] has emerged as an active research line focused on designing cryptographic primitives, schemes, and protocols tailored to constrained devices as sensor nodes in WSN or other IoT devices, for example, RFID tags [5]. For the case of PKC, elliptic curve cryptography (ECC) has been considered one of the most efficient realizations well suited for constrained environments in the IoT [6].

Application-specific integrated circuits (ASICs) were the first targets in LWC [4, 7]. However, reconfigurable logic circuits, specifically field programmable gateway arrays (FPGAs), are being more popular to implement compact/low-area hardware accelerators for cryptography algorithms, with attractive advantages for the IoT domain [8]. At the beginning, FPGAs were frequently used as devices for rapid prototyping of cryptographic algorithms, but now they are commonly used as final product platforms [9]. Furthermore, FPGAs are not only used as single parts of embedded systems but rather as system-on-chip (SoC) platforms for implementing complete applications [10]. Modern, commercial FPGA devices contain not only programmable hardware resources but large functional blocks, such as high-speed multipliers, embedded multiport memories, and even programmable processor cores, thus enabling hardware/software codesigns where the critical parts of algorithm, protocol, or application are accelerated with custom designs implemented in the available programmable hardware, and the rest of the application is executed by the general purpose processors. The main advantage of FPGAs is reconfigurability since, for example, a whole system could be upgraded (or partial reconfigured) [7].

Recent works propose FPGAs as the most attractive candidates to a large range of IoT applications because of their high energy efficiency and low cost, for example, for IoT machine learning [11], IoT neural networks [12], IoT vehicle monitoring systems [13], IoT security (cryptography) [14], and among other applications. Not only research papers propose FPGAs as hardware modules for IoT scenarios but also FPGA vendors are producing devices with specific features for IoT development [15].

Contribution: in this work, we aim at approaching low-area hardware engine to ECC for IoT security, suitable for being included as a building block in FPGA-based sensor nodes for IIoT or MIoT. We aim at providing one of the most compact FPGA hardware accelerator for the scalar multiplication in binary standard curves, the most time consuming operation, and the core of ECC cryptographic schemes such as encryption, digital signatures, and key establishment. To achieve compactness, a novel digit-digit binary finite field multiplier is proposed and used as the basic building block of the proposed ECC accelerator. Under this approach, the operands are processed one digit at a time in an iterative way, but exploiting the parallelism at the algorithmic level and reusing hardware resources as much as possible. The sequence of field operations in the algorithm for scalar multiplication is carefully scheduled to reduce the number of field multiplier cores (two) and memory blocks (eight). While the field multipliers are implemented using standard FPGA logic, memories are taken from the ones available in modern FPGAs. Due to the digit-digit computation approach, an efficient data memory management is designed to reduce the number of memory block. This way, with only the eight memory blocks, the several field multiplications in a single point addition are correctly computed, and at the same

time, those same memories serve to keep the progress of the scalar multiplication computation. The novel hardware design presented in this work was validated under a hardware/software implementation of elliptic curve Diffie-Hellman (ECDH) key exchange protocol, tailored to the MicroZed FPGA prototyping board, recommended for IoT industrial applications. Under this setting, which is very common in an FPGA IoT application, the execution of ECDH outperforms the software counterpart, implemented using the MIRACLE library and runs in the embedded Cortex A9 processor in the MicroZed. Our hardware architecture, compared with state-of-the-art similar approaches in terms of area, only requires up to 16% of FPGA hardware resources, thus being the most compact FPGA-based hardware architecture for computing scalar multiplications in ECC defined over binary fields. Compared to the software reference implementation, our design is 17 times faster.

The rest of this brief is organized as follows: *Materials and Methods* discusses the preliminaries of scalar multiplication in binary elliptic curves and the Montgomery López-Dahab algorithm for scalar multiplication. This section also describes related works and the proposed hardware design. *Results and Discussion* presents the experimental results and comparisons with state-of-the-art works, followed by concluding remarks in the *Conclusion*.

2. Materials and Methods

First, we provide the mathematical concepts and foundations that are the basis to construct the FPGA-based ECC cryptoengine. First, we present the basis of elliptic curves and groups from which the scalar multiplication is defined. Scalar multiplication is critical because the proposed hardware cryptoengine is precisely to speed up this costly operation and the core of higher operations for security applications such as encryption and digital signatures. Finally, the section concludes discussing the method to compute scalar multiplications on binary elliptic curves. This algorithm is realized by the proposed FPGA-based ECC cryptoengine.

2.1. Elliptic Curves and Its Use in Cryptography. Since invented independently by Miller [16] and Koblitz [17], elliptic curve cryptography (ECC) has received a lot of attention in the academy and industry. Elliptic curves and their properties have enabled also other types of cryptography relevant for the IoT (in wireless sensor networks), for example, identity-based encryption (IBE) [18] and attribute-based encryption [19]. With the advent of the IoT, mainly plagued by intelligent object with restricted computing and resources capabilities, ECC is becoming one of the promising approaches to provide security services in that computing paradigm [6].

An elliptic curve $\mathbb{E}$ over a finite field $\mathbb{F}_q$ is defined by Eq. (1).

$$\mathbb{E} : y^2 + a_1xy + a_3y = x^3 + a_2x^2 + a_4x + a_6, \quad (1)$$

where $a_1, a_2, a_3, a_4, a_6 \in \mathbb{F}_q$. The (x, y) pairs satisfying $\mathbb{E}$, together with a special point named point at infinity O, form a group $\mathbb{G}$ with point addition as the group operation. $\mathbb{G}$ is a cyclic group with prime order n where the discrete logarithm problem is defined and on which ECC is founded.

It is well known that binary extension fields ($q = 2^m$) are very attractive for defining ECC. An element in $\mathbb{F}_{2^m}$ is the bit vector $(a_{m-1}, a_{m-2}, \cdots, a_0)$ that in polynomial basis represents the $(m-1)$-degree polynomial $a_{m-1}x^{m-1} + a_{m-2}x^{m-2} + \cdots a_0$, with a_i in {0,1}. Arithmetic in $\mathbb{F}_{2^m}$ in polynomial basis is polynomial arithmetic with reduction modulo, which is an irreducible polynomial of degree m, $F(x)$. The arithmetic in $\mathbb{F}_{2^m}$ is carry free and more suitable for hardware implementations.

2.2. Scalar Multiplication in Elliptic Curves. Scalar multiplication in $\mathbb{E}(\mathbb{F}_q)$ denoted as $Q = kP$ with $Q, P \in \mathbb{G}$ and $k \in [1, n-1]$ is the main and most time-consuming operation in any ECC scheme (encryption, digital signature, keys exchange, etc). Q is computed by k-times point addition operations of P with itself [20]: $Q = kP = \underbrace{P + P + \cdots + P}_{k\text{–times}}$.

The complexity of kP is in terms of the operations in $\mathbb{F}_q$. Given a large integer k and a point P in $\mathbb{G}$, it is easy to compute $Q = kP$. On the contrary, the elliptic curve discrete logarithm problem (ECDLP) is the problem that given the point P and Q in $\mathbb{G}$, to find the scalar k. For an enough large n, ECDLP becomes hard to solve. Most of the state-of-the-art works related to ECC have been focused on the efficient implementation of scalar multiplication [6], which is a condition for efficient ECC implementation.

The Lopez-Dahab Montgomery PM algorithm [21], shown in Algorithm 1, has been commonly used for the kP computation because it is side-channel attack-resistant, suitable for parallelization and low resource friendly. In this work, we use the Lopez-Dahab algorithm for implementing for the first time the most compact FPGA-based hardware architecture for computing kP in binary elliptic curves, $E(\mathbb{F}_{2^m})$.

The main operations in Algorithm 1 are addition, multiplication, and squaring in $\mathbb{F}_{2^m}$. Consider the fields recommended by NIST for practical ECC, with $m = 233$ and $m = 409$. For $m = 409$, 2.2 will have a cost of 1227 field additions, 2454 field multiplications, and 2454 field squarings over $\mathbb{F}_{2^m}$, being field multiplication the most time-consuming operation.

The Lopez Dahab's method for scalar multiplication in ECC is considered as the most suitable method when targeting low computing powered devices [22]. The elliptic curve point is represented in projective coordinates. At the beginning, the elliptic curve point P in affine coordinates (x, y) is converted to its projective representation (X, Y, Z). Algorithm 1 uses the x-coordinate only for point representation so storage resources can be saved (line 5). With this setting, costly field inversions are avoided in each group (curve level) operation. Only one field inversion is required for coordinate conversion from projective to affine at the end of the main loop (line 13). Algorithm 1 is time-constant and resistant to some side-channel attacks such as simple power analysis (SPA).

$E(\mathbb{F}_{2^m})$
Require: $k \geq 0$
Require: $P = (x, y) \in E(\mathbb{F}_{2^m})$
1: **function** MONTGOMERYk, P
2: **if**$k = 0$ or $x = 0$**then**
3: **return**$(0, 0)$.
4: **end if**
5: $P_1(X_1, Z_1) \leftarrow (x, 1)$; $P_2(X_2, Z_2) \leftarrow (x^4 + b, x^2)$
6: **for** i from $l - 2$ downto 0**do**
7: **if**$k_i = 1$**then**
8: $P_1 \leftarrow$ MaddP_1, P_2; $P_2 \leftarrow$ Mdouble(P_2)
9: **else**
10: $P_2 \leftarrow$ MaddP_2, P_1; $P_1 \leftarrow$ Mdouble(P_1)
11: **end if**
12: **end for**
13: **return**$Q =$ Mxy(P_1, P_2, P)
14: **end function**
1: **procedure** MADDP_1, P_2
2: $Z_3 \leftarrow (X_1 Z_2 + X_2 Z_1)^2$; $X_3 \leftarrow xZ_3 + X_1 X_2 Z_1 Z_2$
3: **return**$P_3(X_3, Z_3)$
4: **end procedure**
1: **procedure** MDOUBLEP_1
2: $Z_2 \leftarrow X_1^2 Z_1^2$, $X_2 \leftarrow X_1^4 + bZ_1^4$
3: **return**$P_2(X_2, Z_2)$
4: **end procedure**
1: **procedure** GXYP_1, P_2
2: $x_q \leftarrow X_1 Z_1^{-1}$
3: $Y_{int} \leftarrow (X_1 + xZ_1)(X_2 + xZ_2) + (x^2 + y)Z_1 Z_2$
4: $y_q \leftarrow (x + x_q) Y_{int} (xZ_1 Z_2)^{-1} + y$
5: **return**$Q(x_q, y_q)$
6: **end procedure**

Algorithm 1: Montgomery scalar multiplication [21].

2.3. Related Work. Being $kP$the core operation in ECC cryptographic schemes, that operation has been the main target for hardware accelerations; however, few works have approached low-area designs compared to those trying to achieve the maximum performance. However, for the devices used in the IoT, generally sensor nodes, lightweight realizations of cryptography are better preferred to efficiently use the available computing and power resources in the sensor nodes [23].

The computation of kP implies to execute an scalar multiplication algorithm, being Algorithm 1 one of the most recommended. At each iteration, curve (group) arithmetic is executed, either point addition or point doubling, each implying several finite field operations. So, operations in groups and finite fields are critical for public key cryptography as in elliptic curve cryptography (ECC). An efficient implementation of kP requires an efficient implementation of finite field operations, being multiplication and inversion the most time consuming field operators. Field inversion can be efficiently realized through several field multiplications; consequently, hardware field multiplier has been studied as the main core to compute kP.

In the case of $\mathbb{F}_{2^m}$, there are three main families of algorithms to compute a field multiplication $A(x) \times B(x)$ mod $F(x)$: full-parallel, bit-serial, and digit-serial [24]. The full-parallel approach is the most costly in terms of area usage but is the fastest while the bit-serial approach is generally the most compact but its slower. The digit-serial approach allows a trade-off between computation time and area usage.

Related works are discussed in this section, based on the type of multiplier being used (bit-serial, digit-serial), computing approach (LSE, MSE), the implementation platform (FPGA type), the finite field size, and implementation results in terms of time and area (FPGA slices). Note that our contribution is on the multiplier being used and in the computing approach (digit-digit). This approach has not been explored, and we present for the first time an FPGA accelerator for ECC based on such approach.

Digit-serial and bit-serial approaches to field multiplication are iterative algorithms that process one of the operands in the multiplication from right-to-left (MSE) or from left-to-right (LSE). At each iteration, the partial results need modular reduction. Bertoni et al. [25] presented an easy way to perform modulo reduction when partial results have coefficients with powers greater than $m - 1$ (e.g., a^m). Beuchat et al. [24] surveyed some of the most representative $\mathbb{F}_{2^m}$ implementations using MSE and LSE algorithms (including implementations presented in [25]).

Digit-serial implementations (with digit size D) require $\lceil m/D \rceil$ iterations using $(m-1)$-degree partial results [26]. However, in [27], it is proposed to use $(m + D - 1)$-degree partial results to improve computation performance at the cost of one extra iteration, requiring $m + 1$ iterations to compute multiplication over $\mathbb{F}_{2^m}$. The digit-serial algorithm proposed in [25] requires $m + 1$ iterations and keeps $(m + D - 1)$-degree partial results to improve computation performance. Beuchat [24] concluded that the MSE first approach requires less hardware and offers higher throughput than LSE. In [28], the reduction steps are performed separately. It is stated that for a finite field generated by irreducible polynomials $F(x)$ (NIST [29]), reduction can be performed by a set of *xor* operations [30, 31]. [28] is considered only the multiplication step, implemented in a digit-serial approach. A digit $D = 16$ is proposed since in most cases, 16-bit words give better results.

In [32], it is used a LSE digit-serial multiplier; however, a digit size of one bit (bit-serial) resulted the most compact version. [33] is proposed a systolic hardware architecture to compute multiplication/inversion in the same hardware. Furthermore, an arithmetic unit is constructed that can perform all $\mathbb{F}_{2^m}$ arithmetic operations required in elliptic curve cryptography. [34] is presented for the first time a digit-digit $\mathbb{F}_{2^m}$ multiplier under a MSE basis. Operands, modulus, and partial results are partitioned in digits and processed one digit at a time. The main advantage compared to digit-serial or bit-serial implementations is that operands and partial results can be stored in BRAMs instead of shift registers which saves standard logic (slices). However, the multiplier presented is designed and evaluated as a standalone module which is hard to directly use in a kP engine.

Table 1 summarizes the most relevant works for $\mathbb{F}_{2^m}$ multiplication in FPGA, the main algorithms used, and the area/-time results. Table 2 shows some of the most representative works of hardware designs for kP computation in the

Table 1: Hardware approaches for $\mathbb{F}2m$ multipliers.

Ref.	Field	Target	Algorithm	Approach	Slices	Time (ns)
[24]	$F(2233)$	Spartan 3	MSE	Digit-serial	3458	58.0
[24]	$F(2233)$	Spartan 3	LSE	Digit-serial	3504	62.0
[24]	$F(2409)$	Spartan 3	MSE	Digit-serial	5406	153.0
[28]	$F(2233)$	Virtex 6	Schoolbook method	Digit-serial	1643 (LUTs)	802.4
[32] (d=1)	$F(2233)$	Virtex 5	LSE	Digit-serial	714 (LUTs)	415.0
[32] (d=16)	$F(2233)$	Virtex 5	LSE	Digit-serial	2351 (LUTs)	35.0
[34]	$F(2233)$	Spartan 3	MSE	Digit-digit	406	219.0
[33]	$F(2163)$	Virtex II	M-I algorithm	Systolic array	1399	

Table 2: Hardware approaches for kP in FPGAs.

Ref.	Field	Target	Mult. algorithm	Approach	Slices	Time
[4]	$F(2193)$	ASIC	MSE	Digit-serial	17723 GE	41.70 ms
[32]	$F(2233)$	Virtex 5	MSE	Digit-serial	6487	19.89 μs
[38]	$F(2233)$	Virtex 7	MSE	Digit-serial	2647	16.01 μs
[39]	$F(2163)$	Spartan 3	LSE	Bit-serial	3383	2.23 ms
[40]	$F(2193)$	Spartan 3	Comba wxw	Digit-serial	473	125.00 ms
[37]	$F(2233)$	Kintex 7	MSE	Bit-serial	3016	2.66 ms
[41]	$F(2163)$	Virtex 5	Karatsuba	Bit-parallel	3789	10.00 μs

hardware. Most of the reported works use the bit-serial or digit-serial approach to implement hardware $\mathbb{F}_{2^m}$ operators. However, hardware resources required in these approaches depend directly on the operands size (field size m), because even when one of the operands is iteratively processed, the other one is processed in parallel.

The bit-serial approach requires small amount of hardware resources compared to the digit-serial or full-parallel approach, but for large operands, even using the bit-serial approach requires a considerable amount of hardware resources (slices). However, some recent works already proposed using a digit-digit approach, for example, [34, 35]. The main drawback with the multiplier presented in [34] is the use of shift registers to store partial results and the infeasibility of using such design for practical kP engine and for [35] is to fit the digit sizes to FPGAs embedded DSP multipliers.

In order to reduce area requirements and achieve a compact design well suited for IoT applications, the approach in this work to construct a hardware kP accelerator follows the digit-digit computation approach and makes use of multipliers and memory blocks embedded in most of the FPGAs to save FPGA standard logic. By implementing a strategy for reusing memory blocks, critical for the iteratively processing of the digit-digit approach, considerable area resources are saved but retaining the advantage of processing iteratively both operand in the multiplication and not only one as in the digit-serial or bit-serial approaches. Additionally, since memory blocks are bigger than operands, it is proposed to used part of the available memory blocks to store control signals thus (microprogramming) avoiding logic to implement a state machine for control.

2.4. Novel Digit-by-Digit Elliptic Curve Point Multiplication Hardware Architecture. The proposed ECC engine, suitable for FPGA-based sensor nodes in the IoT, is constructed following a layered-based approach. The low level is the $\mathbb{F}_{2^m}$ arithmetic, where field multiplication is the main operation to be optimized in terms of area resources. Next, using the $\mathbb{F}_{2^m}$ multiplier as a building block in the high layer is the curve arithmetic, consisting in the optimized realization of Algorithm 1 in terms of area resources, where the $\mathbb{F}_{2^m}$ multiplier is used to compute each of the point additions (lines 8 and 10). At this level, the $\mathbb{F}_{2^m}$ multiplier is used to realize field inversion and field squaring required in the addition and double point operations. In both layers, the proposed design methodology takes advantage of block RAMs (BRAMs) embedded in modern FPGAs to store the operands, partial, and final results, reusing the BRAMs as much as possible, using a carefully field operation scheduling, and memory management strategy.

2.4.1. Field Arithmetic. Arithmetic in $\mathbb{F}_{2^m}$ is done using polynomial basis. Under this representation, each element in the field is an $(m-1)$-degree polynomial $A(x)$ over the field $\mathbb{F}_2$. The two $\mathbb{F}_{2^m}$ binary operators are addition and multiplication with reduction modulo which is an irreducible polynomial $F(x)$ of degree m. Field addition is the bit-wise XOR operation of coefficients (carry free, no reduction needed), a cheap

operation when implemented in the hardware. Additive inverse in $\mathbb{F}_{2^m}$ under polynomial basis is also easy to implement, as for any $A(x)$ in $\mathbb{F}_{2^m}$, $A(x) + A(x) = 0$, with 0 as the neutral addition element (all zero polynomial).

Multiplication and multiplicative inverses (or simply inversion) in $\mathbb{F}_{2^m}$ are more complex operations. Since Algorithm 1 only requires one $\mathbb{F}_{2^m}$ inversion at the end of the computation, field inversion is implemented using the Itho-Tsuji algorithm, by a series of $\mathbb{F}_{2^m}$ multiplications. So, the field multiplier becomes the most critical operation to be carefully implemented in ECC hardware approaches and one of the critical component in our kP engine.

2.4.2. $\mathbb{F}_{2^m}$ Multiplication. In the literature, there are basically three computing approaches for computing field multiplication in the hardware: bit-serial (the most compact design), digital-serial (for area-performance trade-offs), and full-parallel (the fastest but also the costlier solution in terms of area). The most significant element (MSE) and least significant element (LSE) (bit-serial or digit-serial) are the commonly used algorithms to compute multiplications over $\mathbb{F}_{2^m}$.

In this work, we propose a novel digit-digit $\mathbb{F}_{2^m}$ multiplier algorithm well suited to be integrated into a kP engine. The digit-digit computing approach aims at performing better than a bit-serial multiplier, keeps the property of allowing exploring area-performance trade-offs when realized in hardware, and it is not as expensive as a full parallel realization. This is consistent with our design methodology to achieve a compact architecture (simpler datapath) for the kP engine. Details of the digit-digit $\mathbb{F}_{2^m}$ multiplier are presented in Section 2.4.3.

$\mathbb{F}_{2^m}$ multiplication using the digit-digit computing approach was previously suggested in [34]. However, the multiplier design in that work is not suitable for a direct application in a kP engine. The authors in that work only proved the advantages of the digit-digit approach versus the well-known bit-serial and digit-serial multipliers, as a standalone module. However, when that multiplier is considered for realizing the kP operation, several issues must be solved.

Being the multiplier part of a series of operations implied by each point addition operation in the main loop in the kP computation, the main challenge for the digit-digit multiplier is the fact that partial results at each iteration in the digit-digit multiplier and the final result (possibly operated with other values) are the input operand for the same multiplier in next iterations. So, during the digit-digit computation, the multiplier must keep its operands in memory blocks M_1 and M_2 and progressively stores the partial results in another one M_3. At the end, the results in M_3 should be moved to M_1 or M_2 for further processing (a kP operation requires several $\mathbb{F}_{2^m}$ multiplications), introducing a delay in the kP computation, unless that data movement is done during the computation. So, M_1 or M_2 must act as an input and output memory at the same time. Since a complete kP operation requires several hundreds of multiplications, using the multiplier as proposed in [34] without addressing the previous data memory management issue is totally unpractical.

As it is explained in the next section, the main issue to integrate a digit-digit $\mathbb{F}_{2^m}$ multiplier in the kP engine is to implement an efficient data memory management, ensuring consistency in the correct execution of both the digit-digit field multiplier and the scalar multiplication algorithm. In this work, we present the design of a novel digit-digit $\mathbb{F}_{2^m}$ multiplier that achieves compact designs by optimizing the resources for finite fields defined by trinomials.

2.4.3. Digit-Digit $\mathbb{F}_{2^m}$ Multiplier. Parting from the definition of elements in $\mathbb{F}_{2^m}$, as polynomials of the form $b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots b_0$ with binary coefficients, in this section, we present how the mathematical expression that computes an $\mathbb{F}_{2^m}$ multiplication in a digit-by-digit fashion is derived (from Eq. (2) to Eq. (9)). This expression leads to the specification of the $\mathbb{F}_{2^m}$ multiplier that is the building block of our FPGA-based engine for scalar multiplication in ECC.

An element $B \in F_{2^m}$ of the form $b_{m-1}x^{m-1} + b_{m-2}x^{m-2} + \cdots b_0$ can be represented as the sum of $w = \lceil m/d \rceil$ polynomials (digits) each of d coefficients in $\mathbb{F}_2$ (Eq. (2)).

$$B(x) = \sum_{i=0}^{m-1} b_i x^i = \sum_{i=0}^{w-1} B_i x^{id}, B_i = \sum_{j=0}^{d-1} b_{id+j} x^j. \tag{2}$$

So, Eq. (3) expresses the multiplication $C(x) = A(x) \times B(x) \bmod F(x)$ in a digit-serial approach.

$$\begin{aligned} C &= A \times B \bmod F(x) \\ &= \left(A \times \sum_{i=0}^{w-1} B_i x^{di} \right) \bmod F(x) \\ &= AB_0 \bmod F(x) + AB_1 x^d \bmod F(x) \\ &\quad + AB_2 x^{2d} \bmod F(x) + \vdots AB_{w-1} x^{(w-1)d} \bmod F(x). \end{aligned} \tag{3}$$

Let $P^{<i>}(x) = AB_i$, $0 \le i \le w-1$, and the $(d+m-2)$-degree polynomial resulting from the partial product at iteration i in Eq. (3). By parsing elements of B from left-to-right (MSE), C computation at iteration i is determined by recurrence in Eq. (4):

$$C^{<0>} = 0, \tag{4}$$

$$C^{<i+1>} = x^d \left(C^{<i>} \bmod F(x) \right) + P^{<w-1-i>}(x), \tag{5}$$

where polynomial $x^d(C^{<i>} \bmod F(x))$ has the most degree $(d+k-1)$, while $P^{<i>}$ is of degree $(d+k-2)$. After w iterations, the polynomial $C^{<w-1>}$ of degree $(d+k-1)$ needs reduction. By introducing an extra iteration with $B_{-1} = 0$ and $P^{<-1>} = 0$, $C^{<w>} = x^d(C^{<w-1>} \bmod F(x))$ is the result. The x^d term in this last expression can be easily reduced modulo $F(x)$ by only discarding the digit $C_0^{<w>}$.

Being $F(x)$ an m-degree polynomial, $F(x) = x^m + \sum_{i=0}^{m-1} f_i \alpha^i$. So, $x^m \bmod F(x) = \sum_{i=0}^{m-1} f_i \alpha^i = g(x)$, a polynomial of degree g with $g < m$. Thus, elements x^{m+t} with $t \le m-1-g$ can be reduced using equivalence $x^{m+t} \bmod F(x) = g(x)x^t$.

Degree of $C^{<i+1>}$ from Eq. (5) (after $C^{<i>}$ reduction) is at most $(d+m-1)$. This polynomial becomes the $C^{<i>}$ polynomial to be reduced in the next iteration $(C^{<i>} \bmod F(x))$. So,

at each iteration $i+1$, it is required to reduce the d-terms x^j of $C^{<i>}$, $m-1 < j \leq d+m-1$. By using the previous assumption for polynomial reduction being $F(x)$ a trinomial, the reduction in Eq. (5) can be defined as in Eq. (6).

$$\begin{aligned} C^{<i>} \bmod F(x) &= \sum_{i=0}^{m-1} c_i x^i + \left(\sum_{i=m}^{m-1+d} c_i x^i \right) \bmod F(x) \\ &= \sum_{i=0}^{m-1} c_i x^i + \left(\sum_{i=0}^{d-1} c_{m+i} x^j g(x) \right) \bmod F(x) \\ &= C_m^{<i>}(x) + C_d^{<i>}(x) \times g(x). \end{aligned} \tag{6}$$

This way, $C^{<i>}$ is partitioned in two polynomials $C_m^{<i>}(x)$ and $C_d^{<i>}(x)$ of degree $m-1$ and d, respectively. The partial multiplication $C_d^{<i>} \times g(x)$ will not require modular reduction if $d+g<m$. So, Eq. (5) can be rewritten as in Eq. (7).

$$C^{<i+1>} = \alpha^d \left(C_m^{<i>} + C_d^{<i>} \times g(x) \right) + P^{<w-1-i>}. \tag{7}$$

Under the digit-digit computation approach, the polynomial $C_m^{<i>}$, $g(x)$, and A is represented in $w = \lceil m/d \rceil$ digits. Since the B_i degree is $d-1$, the $P^{<i>}$ computation can be achieved iteratively, taken digit B_i and iterating through A digits. Taking B_i as a constant, $P^{<i>}(x) = A(x) \times B_i(x) = \sum_{j=0}^{w-1} (A_j \times B_i) x^{jd} = \sum_{j=0}^{w-1} P_j^{<i>} x^{jd}$. With this new notation, the first term in Eq. (4) can be rewritten as in Eq. (7).

$$\begin{aligned} x^d \left(C_m^{<i>}(x) + C_d^{<i>}(x) \times g(x) \right) &= \sum_{j=0}^{w-1} C_j^{<i>} x^{jd+d} + C_d^{<i>}(x) \times \sum_{j=0}^{w-1} G_j x^{jd+d} \\ &= \sum_{j=0}^{w-1} \left(C_j^{<i>} + C_d^{<i>}(x) \times G_j \right) x^{jd+d} \\ &= \sum_{j=0}^{w-1} R_j^{<i>} x^{jd+d}. \end{aligned} \tag{8}$$

Once $P^{<i>}$ and $R^{<i>}$ are expressed to be processed in an iterative way one digit at a time, Eq. (7) can be rewritten in a notation that leads to an iterative, digit-by-digit computation of each partial product of $\mathbb{F}_{2^m}$ multiplication, given by Eq. (9).

$$C^{<i+1>} = \sum_{j-0}^{w-1} \left(R_j^{<i>} \alpha^{jd+d} + P_j^{<i>} \alpha^{jd} \right). \tag{9}$$

At each iteration, values $P_j^{<i>}$ and $R_j^{<i>}$ can be computed in a parallel way. For the sake of clarity about the computations in Eq. (9), the sum of digits $P_j^{<i>}$ and $R_j^{<i>} x^d$ can be expressed as a single variable $S_j^{<i>}$. This new variable $S_j^{<i>}$ is $(d+d+d)$ bits in size as shown in Figure 2.

With all these considerations, the proposed algorithm for computing multiplication over $\mathbb{F}_{2^m}$ is presented in Algorithm 2.

2.4.4. Digit-Digit $\mathbb{F}_{2^m}$ Multiplier Hardware Architecture. To achieve compactness, in this work, we propose the realization in hardware of Algorithm 2 in its simplest form. The hardware architecture only requires one partial $d \times d$ multiplier and is optimized for binary fields defined by a trinomial. The NIST and other compliant standards have recommended trinomials for binary fields, for example, $F(x) = x^{409} + x^{87} + 1$ and $F(x) = x^{233} + x^{74} + 1$.

If the 233-degree trinomial is used, $g(x) = x^{74} + 1$ is used for the reduction step. So, if $d = 74$ (digit size) is used, when a digit j of g(x) (G_j) is read, only the two first digits will have a value of 1, when $j > 1$ digit G_j will be always 0. In this case, the partial multiplier that computes $C_d^{<i>}(x) \times G_j$ always computes a multiplication of the form $(C_d^{<i>}(x) \times 1)$ or $(C_d^{<i>}(x) \times 0)$ which can be implemented only with an "and" gate. In conclusion, when a trinomial of the form $x^m + x^k + 1$ is used, it is possible to define the digit size $d = k$. In this case, the partial multiplier that computes $C_d^{<i>}(x) \times G_j$ can be implemented using only a multiplexer as it is shown in Figure 3.

2.4.5. Curve Arithmetic. The hardware for elliptic curve scalar multiplication is guided by the execution of Algorithm 1, which is based on the iteratively call to point addition functions *Madd* and *Mdouble*.

Figure 4 shows the required operations at each iteration of Algorithm 1 and the underlying $\mathbb{F}_{2^m}$ operations (denoted by circles). After each $\mathbb{F}_{2^m}$ operation, the figure also shows the memory where the intermediate values are stored. For example, the memory $X11$ stores the first field operation $X1 \times Z2$ in the point addition operation. While five $\mathbb{F}_{2^m}$ multiplications are needed to compute a single *Madd* operation, six $\mathbb{F}_{2^m}$ multiplications are required for *Mdouble*.

The schedule of field operations shown in Figure 4 considers only the use of four memories to compute the complete *Madd* function, by reusing the memory blocks properly. For the case of *Mdouble*, also four memories are enough. The memories are alternatively used as shown in the figure to act as the repository for the input parameters to a field multiplier/adder or as the repository for the multiplication/addition result. We stress again the fact that a proper data memory management must be implemented to avoid the delays induced by moving data from the result memory to the input parameter memory in the chained $\mathbb{F}_{2^m}$ operations.

Since in Algorithm 1, only the X and Z coordinates of elliptic curve points in projective representation are used, and each point $P(X, Z)$ is stored in two BRAMs, one for the X and the other for the Z coordinate. In Figure 4, the memories for the points P_1 and P_2 are represented by the variables X_1, X_2, Z_1, Z_2.

For *Madd*, let us consider the first multiplication $X_1 \times Z_2$ stored in $X11$ and the second multiplication $X_2 \times Z_1$ stored in $Z11$. Both multiplications can be done in parallel, with memories X_1, X_2, Z_1, Z_2 acting as reading memories and $X11$ and $Z11$ acting as the writing memories. For the third multiplication $X11 \times Z11$, memories $X11$ and $Z11$ must switch to act as reading memories, and the result can be stored in Z_1, the memory that initially stored one of the input

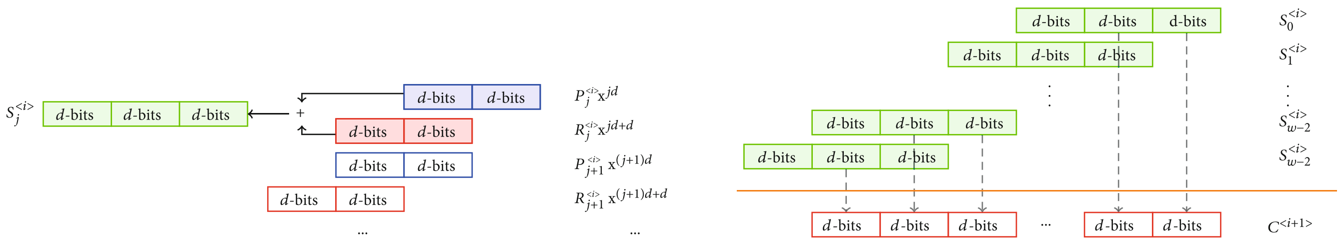

Figure 2: Digit-digit computation of $\mathbb{F}2m$ multiplication.

Require: $A, B, F \in F_2m$ $\triangleright A = \sum_{i=0}^{w-1} \alpha_i \alpha^{iD}$ $\triangleright B = \sum_{i=0}^{w-1} b_i \alpha^{iD}$
$cD \leftarrow 0$
for $i \leftarrow 0$ to $w + 1$ **do**
  $carry \leftarrow 0$
  $s \leftarrow 0$
  for $0 \leftarrow 0$ to w **do**
    $P_i \leftarrow b_{digits-i} \times \alpha_j + 1$
    $R_j \leftarrow c_j + cD \times f_j + 1$
    $s \leftarrow Pi + (R_j \ll d) + carry$
    $c_j \leftarrow s[d-1 \text{ downto } 0]$
    $cD \leftarrow s \gg d$
  end for
  $cD \leftarrow s \gg bitsLastDigit$
end for
$c_w \leftarrow carry$
return c $\triangleright c = A \times B \mod F$ $\triangleright c = \sum_{i=0}^{w-1} c_i \alpha^{iD}$

Algorithm 2: Digit-digit $\mathbb{F}_2m$ multiplier algorithm.

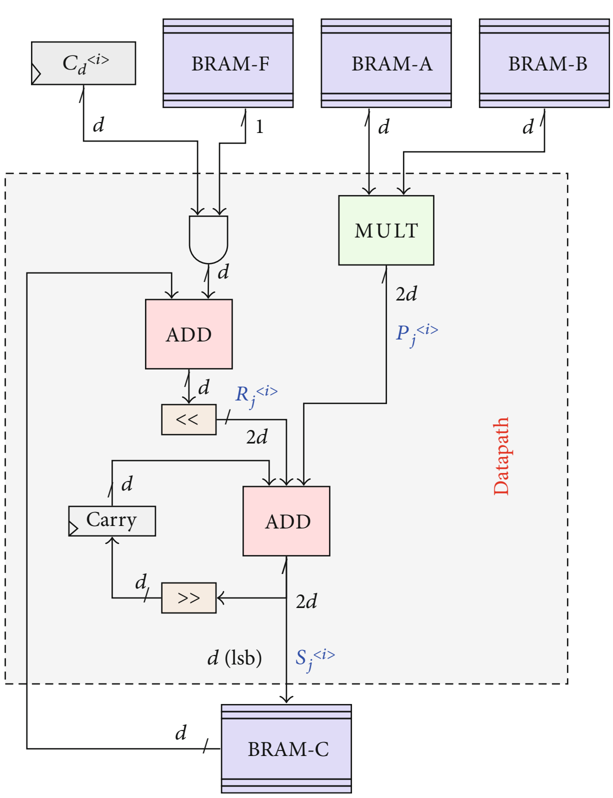

Figure 3: Hardware architecture $\mathbb{F}2m$.

parameters and now acts as a writing memory. As the $\mathbb{F}_{2^m}$ multiplier delivers a result at each stage in point addition, at the same time, it processes the input digits. So, a careful management of the memory is required to avoid latency for data movement for result and input parameter memories. This requirement arises because the result of the field multiplier in an earlier stage becomes the input parameter of later stages.

In the rest of the point addition computation, memories alternate their functionality following the switching strategy of read/write memories. At the end, the final result X_3, Z_3 must be in a memory, that is used in the next iteration at line 6 in Algorithm 1, so that values will reside in one of the four available memories, and input parameters in next iteration in the main loop of Algorithm 1 are adjusted. Memories associated to points $P1(X_1, Z_1)$ and $P2(X_2, Z_2)$ are overwritten with new partial results coming from the *Madd* and *Mdouble* functions.

At line 8 (or also in line 10) in the main loop of Algorithm 1, the memories storing $P_1(X_1, Z_1)$ and $P_2(X_2, Z_2)$ are read memories, and the result is stored finally in memories $P11(X11, Z11)$ and $P22(X22, Z22)$ (see Figure 4). In the next iteration, $P11(X11, Z11)$ and $P22(X22, Z22)$ become P_1 and P_2 input parameters, and the corresponding memories $P_1(X_1, Z_1)$ and $P_2(X_2, Z_2)$ become the storage for the result of the final point addition. So, at the curve level

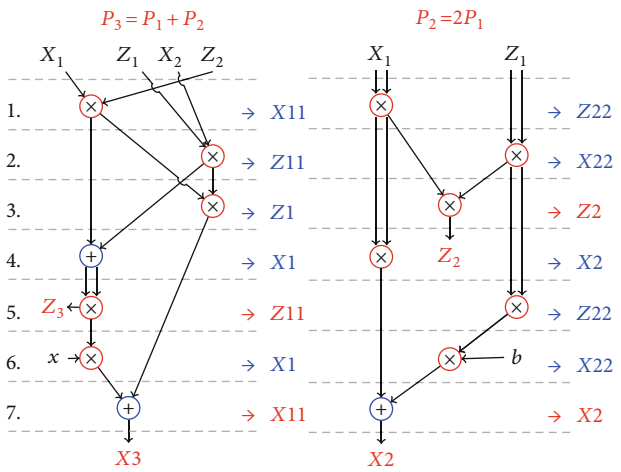

Figure 4: Proposed schedule for point addition/double in $E(\mathbb{F}_2 m)$.

algorithm, the memories are also interchanged in their functionality and properly mapped to the memories for the final results in Figure 4. An extra BRAM is required to store the scalar k.

The building blocks to compute kP as described in Figure 4 are those for field arithmetic operations: addition, multiplication, square, and inversion over $\mathbb{F}_{2^m}$. The square operation is considered easier than multiplication. However, since in this work operands are stored in BRAMs, and reading/writing of operands are performed one digit at a time, it is difficult to take advantage of the optimized algorithm such as the fast reduction algorithm proposed by NIST commonly used in squaring. So, to save hardware resources, this work uses one $\mathbb{F}_{2^m}$ multiplication core to compute square operations. The reusing od the multiplier saves area but increases latency. Also, $\mathbb{F}_{2^m}$ inversion is computed with the Itho-Tsuji algorithm by means of multiplications, squares, and additions in $\mathbb{F}_{2^m}$.

At each iteration of Algorithm 1, *Madd* and *Mdouble* operations can be computed in parallel since there is no data dependency. In this work, we propose to use a $\mathbb{F}_{2^m}$ multiplier in *Madd* and other in *Mdouble* to take advantage of parallelism. In the dataflow for each point addition, the $\mathbb{F}_{2^m}$ multiplier is reused. In addition to the multipliers, one $\mathbb{F}_{2^m}$ adder is also required. The same adder can be used in both the *Madd* and *Mdouble* operations since it is required at different times in each operation.

Although more than one $\mathbb{F}_{2^m}$ multiplier could be added to speed up the kP computation, that approach resulted in extra cost of hardware resources not only because of the area required by the $\mathbb{F}_{2^m}$ multiplier but also for the increased complexity in the control module and additional multiplexers to manage input/output operands to the $\mathbb{F}_{2^m}$ cores.

The entire kP dataflow is managed by a control unit that stimulates the memory blocks for word-based reading and writing and also commands the $\mathbb{F}_{2^m}$ cores (multipliers and adder). The control module waits until each partial multiplication/addition has finished and starts the following required operations with the correct BRAM as input sources.

3. Results and Discussion

The proposed compact hardware ECC design was implemented over the binary fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{409}}$, both defined by an irreducible trinomial. The elliptic curves used were sect233 and sect409, both recommended by NIST and other recognized organizations such as SECG. The target platform was the IoT recommended FPGA board MicroZed, with Xilinx Vivado HLx 2016.4 as the developer tool.

The hardware architecture for scalar multiplication in $E(\mathbb{F}_{2^m})$ was evaluated in a hardware-software codesign of the Diffie-Hellman key exchange elliptic curve (ECDH) version. Let it consider that two FPGA-based sensor nodes [36] A and B agree on an elliptic curve group $\mathbb{G}$ with generator P and order n. Then, each party selects a secret integer, for example, r_A and r_B. Using a kP engine, each party computes public values:

$$\underbrace{Q_A = r_A P}_{\text{Sensor}A} \text{ and } \underbrace{Q_B = r_B P}_{\text{Sensor}B}. \quad (10)$$

Sensor A uses the B's public value to compute $s_1 = r_A Q_B$, and the sensor B uses the A's public value to compute $s_2 = r_B Q_A$. Since s_1 is the same as $s_2 (s = s_1 = s_2)$, s acts as a shared secret key between the sensors A and B, so a secure channel can be established to transport data between the two devices in an encrypted form (for example, using a lightweight block cipher). Indeed, signatures can be generated to authenticate data by using the secret to authenticate a message, using, for example, LightMac. The main complexity in ECDH (as in other ECC-based cryptographic schemes) is the computation of kP.

3.1. Hardware/Software Codesign. Figure 5 shows the proposed hardware-software codesign for the scalar multiplier over $E(\mathbb{F}_{2^m})$, suitable to be realized in an FPGA sensor node. The codesign was realized in the MicroZed board, and the implementation results are shown in Table 3. This is a representative final application under an IoT scenario (IIoT, MIoT) where sensor nodes are deployed using SoC technology: the kP scalar multiplication is executed in FPGA technology coupled to a master general purpose processor that runs the rest of the application logic. The hardware-software codesign required 1809 slices of the FPGA embedded in the MicroZed board running at 62.5 MHz.

Table 3 also compares the time to achieve a scalar multiplication under the hardware/software codesign versus a pure software implementation. This is done to highlight the gain in performance from a hardware approach for the most time-consuming operation in ECC, as in ECDH. For this, we used the MIRACL library for the software implementation of scalar multiplication in the Cortex A9 of the Zynq, also available in the MicroZed board. In this case, we used the same implementation parameters: curve, finite field, size of the finite field, irreducible polynomial, projective coordinates, and the same Algorithm 1 for scalar multiplication.

The hardware-accelerated execution of kP requires 4.13 ms to compute an elliptic curve Diffie Hellman key

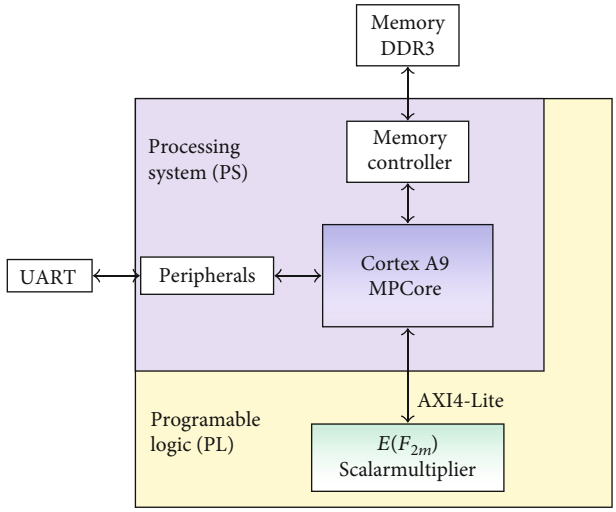

Figure 5: Hardware-software codesign for an FPGA-based sensor node enabled with scalar multiplier engine for curve-based cryptography.

Table 3: ECDH hw-sw codesign in the MicroZed board (z7010).

Size	k	Area (slices)	Freq. (MHz)	Time (ms)	MIRACL (sw) (ms)
233	32	1809	62.5	4.13	70

exchange versus the pure software implementation in the MicroZed with the MIRACL library that requires 70 ms. Thus, our codesign is 17 times faster than the pure software implementation while only requires 36% of the FPGA slices in the MicroZed, leaving 66% of the FPGA's standard logic available for other application requirements in the sensor node. These results show that our design retains the advantages of a hardware implementation by improving the performance at the time that it uses less area resources.

3.2. Comparison with Other Similar FPGA Designs. Table 4 shows a comparison with state-of-the-art works for FPGA scalar multipliers in $E(\mathbb{F}_{2^m})$. In this comparison, we are using the same elliptic curves, finite fields and sizes, and the same irreducible polynomial. A fair comparison is very difficult to achieve due to different FPGA technologies and implementation strategies being used. It is not possible to compare all the works under the same criteria, since some hardware designs exploit the use of embedded blocks such as DSPs or block rams (BRAMs) while others take advantage of the available slices/LUTs. However, this research is focused in lightweight implementations with the goal to use low standard logic resources. So, embedded memory blocks in the FPGAs are exploited to reduce standard reconfigurable logic (slices). The comparison in Table 4 is mainly in terms of FPGA standard logic (slices) reported. Although efficiency and throughput are not the main aims of this research, they are used as reference metrics.

The results presented in [32] are proposed for a digit-serial approach for multiplication and inversion over $\mathbb{F}_{2^m}$, and square and addition over $E(\mathbb{F}_{2^m})$ are computed fully with standard logic in only one clock cycle. Compared to our design, those results are almost ten times better according to efficiency. However, our design uses considerable less area resources. For example, for a digit size of 8, 16, and 32, the required area is 442, 626, and 1170 slices, respectively. In [37], it is presented a hardware architecture for elliptic curve scalar multiplication over $E(\mathbb{F}_{2^m})$ implemented for the NIST-recommended binary fields $\mathbb{F}_{2^{233}}$ and $\mathbb{F}_{2^{283}}$. That scalar multiplier hardware architecture requires 3016 and 4625 slices for the operand size 233 and 283, respectively. Compared to that design, our kP engine for $\mathbb{F}_{2^{233}}$ requires 6.8 times more slices and 2.2 times better efficiency (Mbps/slice). The scalar multiplier over $E(\mathbb{F}_{2^m})$ presented in [38] is better in efficiency than ours, but at a considerable high costs in terms of area usage.

Table 4 shows that most of the works achieve better throughput/efficiency than our proposed hardware design. However, the main aim of these works is to save hardware resources (slices), and this is achieved by sacrificing throughput. According to the obtained results, it is observed that despite the throughput sacrificing, the proposed design achieves significantly better performance than software counterparts while using fewer resources that are similar FPGA designs. The reduction in area resources is a direct result of using a digit-by-digit computing approach in the layered structure of the kP engine, mainly determined by the $\mathbb{F}_{2^m}$ multiplier and the strategy for reusing memory blocks during the iterative processing of operands.

In Figure 6, we show graphically how our design uses considerable fewer standard logic resources from the FPGA, so leaving more logic for other tasks in the upper application layers. In that figure, FPGA resource usage is compared against the works that use FPGA implementation technology, digit-serial approach, and comparable security levels. Note from this figure that our design is scalable in terms of area because a greater security level only impacts latency. This property is only kept with the digit-digit computing approach.

4. Conclusion

We have detailed the design and evaluation of a compact FPGA-based ECC hardware design, well suited for Internet of Things applications, specifically for the Industrial Internet of Things (IIoT) or Internet of Medical Things (MIoT), where sensor nodes can be realized with FPGA technology. The key contributions include a novel digit-digit algorithm for multiplication over $\mathbb{F}_{2^m}$ optimized for fields defined by trinomials and its corresponding compact hardware architecture, which is the main core for constructing a compact hardware design for computing scalar multiplications in binary elliptic curves over $\mathbb{F}_{2^m}$ generated by trinomials, such as the ones recommended by NIST for practical use. We proposed a novel rescheduling of $\mathbb{F}_{2^m}$ operations in the Lopez-Dahab Montgomery algorithm for elliptic curve scalar multiplication that can be computed with only two multipliers and one adder in a digit-digit fashion, thus reducing area requirements for the hardware design. For correctness, we validate our design by a hardware software codesign in the IoT

Table 4: Comparison of scalar multiplication over $E(\mathbb{F}2m)$.

Work	FPGA	m	Cycles	Slices	Freq. (MHz)	Thrg. (kbps)	Efficiency (kbps/slice)
Prop. ($k = 8$)	z7010	233	1553782	442	190.04	28.49	0.064
Prop. ($k = 16$)	z7010	233	408547	626	149.20	85.09	0.136
Prop. ($k = 32$)	z7010	233	128820	1170	135.31	244.75	0.209
Prop. ($k = 8$)	z7010	409	7504232	453	190.94	10.40	0.023
Prop. ($k = 16$)	z7010	409	1926426	653	154.44	32.78	0.050
Prop. ($k = 32$)	z7010	409	511493	1183	132.59	106.02	0.090
[32] ($g = 16$, $d = 2$)	v5	233	8193	3939	263.15	7483.69	1.899
[32] ($g = 8$, $d = 1$)	v5	409	45513	5395	181.81	1633.82	0.030
[37]	k7	233	679776	3016	255.66	87.63	0.029
[37]	k7	283	1395312	4625	251.98	51.10	0.011
[38]	v7	233	5929	2647	370.00	14540.39	5.498
[38]	v7	409	10354	6888	316.00	12482.51	1.812
[41]	v5	163	1396	3513	147.00	17.16	0.004

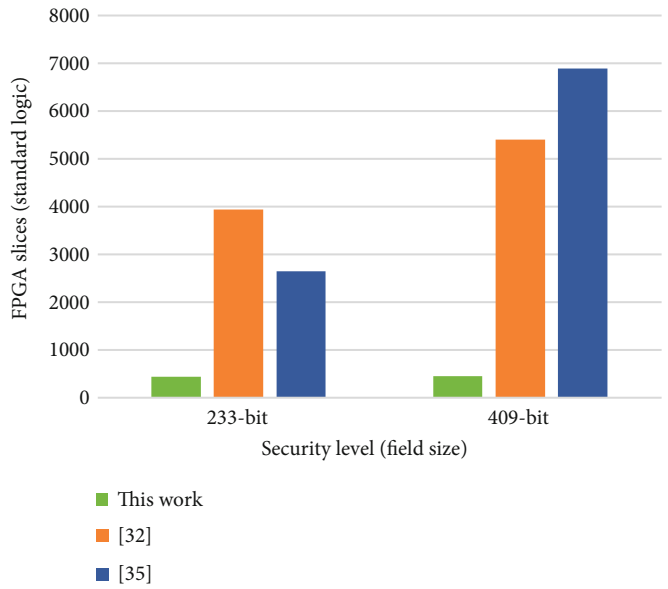

Figure 6: Comparison of area usage for the proposed kP engine.

MicroZed Xilinx FPGA, by executing an instance of the Diffie-Hellman key exchange protocol (ECDH), a common crucial operation in IoT secure sensor nodes networks. To our knowledge, the proposed hardware ECC architecture requires less standard hardware resources (slices) in FPGAs than other works reported to date while takes advantage of memory blocks already available in modern FPGAs. Furthermore, despite of being a compact hardware architecture, it was demonstrated that a considerable acceleration of a representative curve-based cryptographic protocol is obtained compared to a pure software implementation.

Using the proposed ECC accelerator, further work is planned to evaluate the security service costs when implementing ECC-based cryptographic protocols such as digital envelopes and digital signatures in real application scenarios of IoT, IIoT, and MIoT.

Data Availability

Raw data were generated at INAOE Computer Science Department and at Cinvestav Tamaulipas. Derived data supporting the findings of this study are available from the corresponding author MMS on request.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This research was supported by the Fondo Sectorial de Investigación para la Educación, Ciencia Básica SEP-CONACyT,

project number 281565. Also, the research was partially funded by project PN-2017-5814, Conacyt Problemas Nacionales.

References

[1] A. Mosenia and N. K. Jha, "A comprehensive study of security of internet-of-things," *IEEE Transactions on Emerging Topics in Computing*, vol. 5, no. 4, pp. 586–602, 2017.

[2] L. Z. Cai and M. F. Zuhairi, "Security challenges for open embedded systems," in *Engineering Technology and Technopreneurship (ICE2T), 2017 International Conference on*, pp. 1–6, Kuala Lumpur, Malaysia, 2017.

[3] D. Schinianakis, "Alternative security options in the 5g and iot era," *IEEE Circuits and Systems Magazine*, vol. 17, no. 4, pp. 6–28, 2017.

[4] T. Eisenbarth, S. Kumar, C. Paar, A. Poschmann, and L. Uhsadel, "A survey of lightweight-cryptography implementations," *IEEE Design & Test of Computers*, vol. 24, no. 6, pp. 522–533, 2007.

[5] C. Manifavas, G. Hatzivasilis, K. Fysarakis, and K. Rantos, "Lightweight cryptography for embedded systems – a comparative analysis," *Data Privacy Management and Autonomous Spontaneous Security*, 2014, pp. 333–349, Springer, Berlin Heidelberg, 2014.

[6] C. A. Lara-Nino, A. Diaz-Perez, and M. Morales-Sandoval, "Elliptic curve lightweight cryptography: a survey," *IEEE Access*, vol. 6, pp. 72514–72550, 2018.

[7] P. Yalla and J. P. Kaps, "Lightweight cryptography for fpgas," in *2009 International Conference on Reconfig urable Computing and FPGAs*, pp. 225–230, Quintana Roo, Mexico, 2009.

[8] A. Diaz-Perez, M. Morales-Sandoval, and C. Lara-Nino, "Use of FPGAs for enabling security and privacy in the IoT: features and case studies," in *FPGA Algorithms and Applications for the Internet of Things, chapter 2*, P. Sharma and R. Nair, Eds., pp. 26–45, IGI Global, 2020.

[9] G. Xu, Z. Chen, and P. Schaumont, "Energy and performance evaluation of an fpga-based soc platform with aes and present coprocessors," in *Embedded Computer Systems: Architectures, Modeling, and Simulation*, M. Berekovic, N. Dimopoulos, and S. Wong, Eds., pp. 106–115, Springer, Berlin, Heidelberg, 2008.

[10] H. Abdelkrim, S. Ben Othman, and S. Ben Saoud, "Reconfigurable soc fpga based: Overview and trends," in *2017 International Conference on Advanced Systems and Electric Technologies*, pp. 378–383, Hammamet, Tunisia, 2017.

[11] X. Zhang, A. Ramachandran, C. Zhuge et al., "Machine learning on fpgas to face the iot revolution," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pp. 894–901, Irvine, CA, USA, 2017.

[12] C. Hao, X. Zhang, Y. Li et al., "Fpga/dnn co-design: an efficient design methodology for iot intelligence on the edge," in *Proceedings of the 56th Annual Design Automation Conference 2019, DAC '19*, New York, NY, USA, 2019.

[13] S. Wang, Y. Hou, F. Gao, and X. Ji, "A novel iot access architecture for vehicle monitoring system," in *2016 IEEE 3rd World Forum on Internet of Things (WF-IoT)*, pp. 639–642, Reston, VA, USA, 2016.

[14] B. Zhou, M. Egele, and A. Joshi, "High-performance low-energy implementation of cryptographic algorithms on a programmable soc for iot devices," in *2017 IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1–6, Waltham, MA, USA, 2017.

[15] Xilinx Inc, *Microzed industrial iot starter kit*April 2020, http://zedboard.org/product/microzed-iiot-starter-kit.

[16] V. S. Miller, "Use of elliptic curves in cryptography," *Advances in Cryptology—CRYPTO '85 Proceedings*, H. C. Williams, Ed., , pp. 417–426, Springer, Berlin, Heidelberg, 1986.

[17] N. Koblitz, "Elliptic curve cryptosystems," *Mathematics of Computation*, vol. 48, no. 177, pp. 203–209, 1987.

[18] P. Szczechowiak and M. Collier, "Tinyibe: identity-based encryption for heterogeneous sensor networks," in *2009 International Conference on Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP)*, pp. 319–354, Melbourne, VIC, Australia, 2009.

[19] N. Oualha and K. T. Nguyen, "Lightweight attribute-based encryption for the internet of things," in *2016 25th International Conference on Computer Communication and Networks (ICCCN)*, pp. 1–6, Waikoloa, HI, USA, 2016.

[20] Z. U. A. Khan and M. Benaissa, "High-speed and low-latency ecc processor implementation over gf(2^m) on fpga," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 1, pp. 165–176, 2017.

[21] J. López and R. Dahab, "Fast multiplication on elliptic curves over gf(2m) without precomputation," *Proceedings of the First International Workshop on Cryptographic Hardware and Embedded Systems, CHES'99*, , pp. 316–327, Springer-Verlag, London, UK, UK, 1999.

[22] D. Karaklajic, J. Fan, J. Schmidt, and I. Verbauwhede, "Low-cost fault detection method for ecc using montgomery powering ladder," *2011 Design, Automation Test in Europe*, pp. 1–6, 2011.

[23] D. Dinu, Y. Le Corre, D. Khovratovich, L. Perrin, J. Großschädl, and A. Biryukov, "Triathlon of lightweight block ciphers for the internet of things," *Journal of Cryptographic Engineering*, vol. 9, pp. 1–20, 2015.

[24] J.-L. Beuchat, T. Miyoshi, Y. Oyama, and E. Okamoto, "Multiplication over F_pm on fpga: A survey," in *Reconfigurable Computing: Architectures, Tools and Applications*, P. C. Diniz, E. Marques, K. Bertels, M. M. Fernandes, and J. M. P. Cardoso, Eds., pp. 214–225, Springer, Berlin, Heidelberg, 2007.

[25] G. Bertoni, J. Guajardo, S. Kumar, G. Orlando, C. Paar, and T. Wollinger, "Efficient GF(p^m) arithmetic architectures for cryptographic applications," in *Topics in Cryptology — CT-RSA 2003*, M. Joye, Ed., pp. 158–175, Springer, Berlin, Heidelberg, 2003.

[26] C. Shu, S. Kwon, and K. Gaj, "Fpga accelerated tate pairing based cryptosystems over binary fields," *2006 IEEE International Conference on Field Programmable Technology*, 2006, pp. 173–180, Bangkok, Thailand, 2006.

[27] L. Song and K. K. Parhi, "Low-energy digit-serial/parallel finite field multipliers," *Journal of VLSI signal processing systems for signal, image and video technology*, vol. 19, no. 2, pp. 149–166, 1998.

[28] D. Pamula and E. Hrynkiewicz, "Area-speed efficient modular architecture for GF(2m) multipliers dedicated for cryptographic applications," in *2013 IEEE 16th International Symposium on Design and Diagnostics of Electronic Circuits Systems (DDECS)*, pp. 30–35, Karlovy Vary, Czech Republic, 2013.

[29] National Institute of Standards and Technology, *Digital Signature Standard (DSS), Appendix D, Recommended Elliptic Curves for Federal Government Use*, 1999, https://csrc.nist

.gov/csrc/media/publications/fips/186/3/archive/2009-06-25/documents/fips_186-3.pdf.

[30] D. Hankerson, A. J. Menezes, and S. Vanstone, *Guide to Elliptic Curve Cryptography*, Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[31] D. Pamula, *Arithmetic operators on GF(2m) for cryptographic applications: performance - power consumption - security tradeoffs, [Ph.D. thesis]*, Université Rennes 1, 2012, https://tel.archivesouvertes.fr/tel-00767537.

[32] G. D. Sutter, J. Deschamps, and J. L. Imana, "Efficient elliptic curve point multiplication using digit-serial binary field operations," *IEEE Transactions on Industrial Electronics*, vol. 60, no. 1, pp. 217–225, 2013.

[33] A. P. Fournaris and O. Koufopavlou, "Low area elliptic curve arithmetic unit," in *2009 IEEE International Symposium on Circuits and Systems*, pp. 1397–1400, Taipei, Taiwan, 2009.

[34] M. Morales-Sandoval and A. Diaz-Perez, "Area/performance evaluation of digit-digit GF(2^k) multipliers on fpgas," in *23rd International Conference on Field programmable Logic and Applications*, Porto, Portugal, 2013.

[35] I. San and A. Nuray, "Improving the computational efficiency of modular operations for embedded systems," *Journal of Systems Architecture*, vol. 60, no. 5, pp. 440–451, 2014.

[36] B. Bengherbia, M. O. Zmirli, A. Toubal, and A. Guessoum, "Fpga-based wireless sensor nodes for vibration monitoring system and fault diagnosis," *Measurement*, vol. 101, pp. 81–92, 2017.

[37] M. S. Hossain, E. Saeedi, and Y. Kong, "High-speed, area-efficient, fpga-based elliptic curve cryptographic processor over nist binary fields," in *2015 IEEE International Conference on Data Science and Data Intensive Systems*, pp. 175–181, Sydney, NSW, Australia, 2015.

[38] Z. Khan and M. Benaissa, "Throughput/area-efficient ecc processor using montgomery point multiplication on fpga," *IEEE Transactions on Circuits and Systems II: Express Briefs*, vol. 62, no. 11, pp. 1078–1082, 2015.

[39] W. Wei, L. Zhang, and C. Chang, "A modular design of elliptic-curve point multiplication for resource constrained devices," in *2014 International Symposium on Integrated Circuits (ISIC)*, pp. 596–599, Singapore, Singapore, 2014.

[40] M. N. Hassan and M. Benaissa, "Low area-scalable hardware/-software co-design for elliptic curve cryptography," in *2009 3rd International Conference on New Technologies, Mobility and Security*, pp. 1–5, Cairo, Egypt, 2009.

[41] S. S. Roy, C. Rebeiro, and D. Mukhopadhyay, "Theoretical modeling of elliptic curve scalar multiplier on lut-based fpgas for area and speed," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, no. 5, pp. 901–909, 2013.