High-Throughput Bilinear Pairing Processor for Server-Side FPGA Applications

Junichi Sakamoto, *nonmember*, Daisuke Fujimoto, *Senior Member, IEEE*, Riku Anzai, Naoki Yoshida, and Tsutomu Matsumoto, *nonmember*

***Abstract*—This study focuses on the acceleration of cryptographic pairing operations on field-programmable gate arrays (FPGAs) for server-side applications. Previous studies on FPGA pairing implementations focused on area efficiency for embedded devices, trying to achieve maximum performance with minimal circuit resources. However, their architectures are likely to be inefficient for server-side applications, where the primary interest is maximum performance when FPGA resources are finished. Their architectures are inefficient for two reasons: low utilization of the digital signal processor (DSP) and low operation frequency. In this study, we propose a high-throughput pairing processor architecture for server-side FPGAs, taking full advantage of DSPs. First, we propose a loop-unrolled modular multiplication algorithm that is suitable for a server-side FPGA. The algorithm shows the highest throughput and area efficiency compared to algorithms from previous studies. Second, we design a pairing processor architecture that embeds the proposed modular multiplier, thus, maintaining its high throughput by supporting redundant adders and interleaved executions. We evaluate BN254 and BLS12_381 pairings on the proposed processor architecture and the evaluation results show that it achieves good throughput that is approximately 2 and 5 times faster than that from previous studies, respectively.**

***Index Terms*—BN254, BLS12_381, FPGA, Multiplier, Montgomery multiplication, Bilinear pairing**

I. Introduction

With the advent of the Internet of Things (IoT) society, the demand of cryptosystem functions is becoming increasingly diverse. For example, for the demand to reduce the amount of data flowing through a network, aggregate signatures [1] that reduce the number of transactions have been proposed. In addition, for the demand to increase data confidentiality, searchable cryptography has been introduced to enable searching in the ciphertext state [2]. Some of these advanced cryptographies are based on an operation known as the (elliptic curve based) bilinear pairing. A cryptographic system based on pairing is known as pairing-based cryptography (PBC). However, the pairing is computationally intensive, typically requiring more than 10,000 modular multiplications on a finite field [3]. Key size update due to recent theoretical attack [4] also increases the computational complexity of the pairing. For PBC to be widely used worldwide, it is important to accelerate the pairing computations. In this study, we focus on PBC for server-side applications such as aggregate signature verification in Sec. II-B) that requires a large number of pairing computations in a batch manner. Field-programmable gate array (FPGA) implementation is a promising option for accelerating cryptographic operations for server-side applications. Server-side FPGAs such as the Xilinx Virtex Ultrascale+ XVU9P used in AWS, Alibaba, and Huawei clouds [5], [6] have significantly huge resources for a single crypto core; therefore, increasing throughput with a multi-core strategy [7] is an important factor. However, existing studies of FPGA-based pairing processors are unsuitable for multi-core strategy in server-side applications for the following two reasons.

a) Utilized resource imbalance: Several types of circuit resources are available in FPGAs, including look-up tables (LUTs), flip-flops (FFs), and, digital signal processors (DSPs). If the utilization of these resources is unbalanced (if the utilization of one of the resources is high), the number of cores in a multi-core implementation will be bounded by that resource, implying that some resources are unused and do not contribute to performance (see Fig. 1). Existing studies of FPGA pairing processors [8], [9], [10], [11], [12], [13], [14] evaluate only single-core performance; thus, do not take into account the utilized resource balance for multi-core performance.

b) Low operating frequency : Existing pairing processors on FPGAs [8], [9], [10], [11], [14], [15] run at 200 to 300 MHz, which is lower than the maximum operating frequency of the FPGAs, 600 to 800 MHz. This is mainly because the existing studies focus on low-latency computation. Whereas server-side applications require high throughput rather than low latency to respond to a large numbers of cryptographic requests.

To solve these problems, this paper (1) introduces a metric, the Slice DSP Ratio (SDR), that maximizes the use of circuit resources on an FPGA, and using the SDR metric, (2) proposes a methodology to maximize the performance of the crypto core for server-side applications.

To implement a cryptographic core on an FPGA, we can primarily use two resources: A logic slice, consisting of LUTs and FFs, and a DSP, a dedicated unit for sum-of-product operations. Balanced utilization of these two resources

Manuscript received April 19, 2021; revised August 16, 2021.

Junichi Sakamoto is with Yokohama National University, Yokohama, Japan, and with National Institute of Advanced Industrial Science and Technology, Tokyo, Japan, (sakamoto-junichi@aist.go.jp)

Daisuke Fujimoto is with Nara Institute of Science and Technology, Nara, Japan, (fujimoto@is.naist.jp)

Riku Anzai is with Yokohama National University, Yokohama, Japan, (anzai-riku-tm@ynu.jp)

Naoki Yoshida is with Yokohama National University, Yokohama, Japan, (yoshida-naoki-jb@ynu.ac.jp)

Tsutomu Matsumoto is with Yokohama National University, Yokohama, Japan, (tsutomu@ynu.ac.jp)

A part of this work was supported by the Cabinet Office (CAO), Cross-ministerial Strategic Innovation Promotion Program (SIP), "Cyber Physical Security for IoT Society," JPNP18015 (funding agency: NEDO). A Part of this work was supported by JSPS KAKENHI Grant Number JP22K17893.

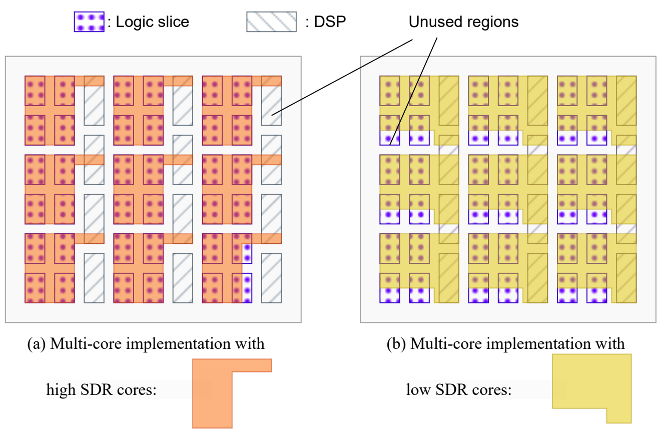

: Logic slice
: DSP
Unused regions
(a) Multi-core implementation with
high SDR cores:
(b) Multi-core implementation with
low SDR cores:

Fig. 1. Abstract images of multi-core implementation with high SDR ($>> 43$) cores (a) and low SDR (< 43) cores (b). (a) has large unused regions including many DSPs, indicating that the FPGA does not achieve maximum performance. (b) can use most of the DSP resources. The remaining unused slices are effectively available for other peripherals such as I/O interfaces.

leads to maximize the performance under multi-core strategy. We introduce the Slice-DSP Ratio (SDR) as a metric for resource utilization imbalance. Xilinx Virtex Ultrascale+ XVU9P, which is a standard server-side FPGA adopted in AWS, Alibaba, and Huawei cloud, provides 295,000 slices and 6,840 DSPs. The closer the SDR of a cryptographic core is to 43[1]$= 295,000/6,840$, the more "balanced" the resource utilization, resulting in high performance due to the full resource utilization as shown in Fig. 1. In previous studies [8], [9], [11], most pairing cores had $SDR > 43$, which is not preferable for maximizing multi-core performance in terms of not taking full advantage of FPGA resources (DSPs).

Our methodology to getting the SDR of the pairing circuit close to 43 is to construct the modular multiplier with as few slices as possible (with low SDR). Since the components of the pairing circuit other than the modular multiplier (e.g., modular adders and sequencers) are composed only of slices, the SDR of the entire pairing circuit is balanced by the low SDR modular multiplier and the high SDR other components. In order to reduce the SDR of the modular multiplier, we propose a new modular multiplication algorithm designed for the DSP of the cloud FPGA. This algorithm is carefully designed to take full advantage of various DSP functions, asymmetric multiplier, ternary post-adder, and pipeline registers, achieving high throughput at low SDR by integrating functions previously implemented in slices into the DSP. The low SDR of the modular multiplier allows us to implement other components for the pairing computation, such as the modular adders, with a large number of logic slices to improve performance. Although adders and sequencers can typically be a performance bottleneck, our pairing processor architecture avoids the performance degradation through deep pipelining and the use of redundant adders that consume a large number of logic slices, resulting in high throughput of the entire pairing computation.

As a demonstration of our proposed method, we show that the proposed modular multiplier achieves a low SDR of less than 22, an operating frequency of over 600 MHz, and at least more than twice the throughput per area compared to previous studies. Furthermore, we evaluate the pairing processor using the proposed modular multiplier on BN254 and BLS12_381 curve pairings. The evaluation results show that can have the good SDRs (46.96 and 41.24) and achieve approximately 2 and 5 times multi-core throughput, compared to those from previous studies. Our source code is available on the web[2].

The remainder of this paper is organized as follows. Section II discusses the preliminaries for this study, including pairing mathematical background and related works. Section III discusses the proposed method and the ways in which it achieved a high performance on server-side FPGAs. Section IV presents the evaluation results of our proposed method on XVU9P FPGA and comparisons with results from previous studies. Finally, Section VI concludes this paper.

II. Preliminaries

A. Pairing

The map below e

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_3 \tag{1}$$
$$e(P, Q) \mapsto R \tag{2}$$

is called (admissible) "pairing" if it satisfies the following three properties:

1) Non-degeneracy: For $\forall P$ ($\forall Q$), if $e(P, Q) = 1$, then $Q = 0$ ($P = 0$).
2) Bilinearity: $e([a]P, [b]Q) = e([b]P, [a]Q) = e(P, Q)^{ab}$.
3) Computability: Efficiently computable in polynomial time.

In the above expression, $\mathbb{G}_1$ and $\mathbb{G}_2$ are represented additive, and $\mathbb{G}_3$ multiplicative. $[a]P$ represents the scalar multiplication of P by an integer a.

Let p be a prime number, E be an elliptic curve defined over $\mathbb{F}_p$, and k is the embedding degree. Our target pairing is the optimal Ate pairing [18], where $\mathbb{G}_1, \mathbb{G}_2$ are defined as subgroups of elliptic curve group $E(\mathbb{F}_p)$, and $\mathbb{G}_3$ is a subgroup of $\mathbb{F}^*_{p^k}$. Although the unique property of the pairing—bilinearity—produces various functionalities in advanced cryptography, the security parameters for using the pairing securely have not been decided yet with solid consensus in the research community. In 2018, Razvan and Sylvain updated the security-bit length for the secure pairing computation [4]. The parameters can be updated again in the future.

Over the optimal Ate pairing [18], the BN curve [19] satisfies a 100-bit security level, if it adopts parameters with 254-bit prime characteristics, while the BLS curve [20] satisfies a 128-bit security level, if it adopts parameters with 381-bit prime characteristics [21].

[1]This is a device-dependent value. We believe that using this value is justified because XVU9P is a de facto standard in cloud FPGAs.

[2]https://github.com/ankoman/HTBPA

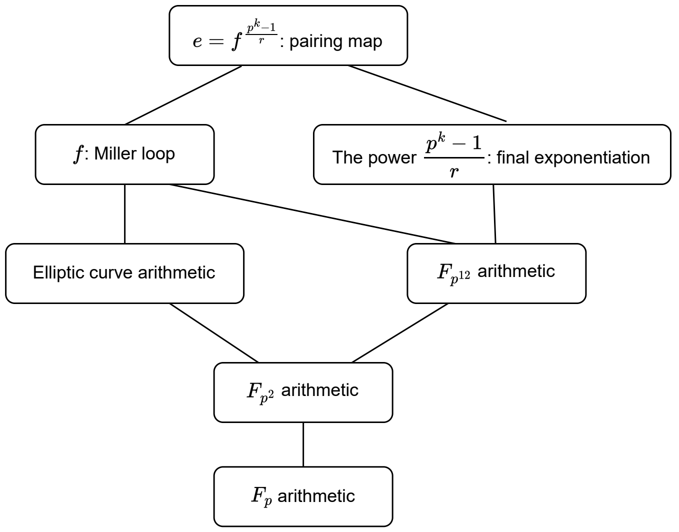

$e = f^{\frac{p^k-1}{r}}$: pairing map
f: Miller loop
The power $\frac{p^k-1}{r}$: final exponentiation
Elliptic curve arithmetic
$F_{p^{12}}$ arithmetic
F_{p^2} arithmetic
F_p arithmetic

Fig. 2. Typical operations required for optimal Ate pairing computation. Hardware architecture suitable for $\mathbb{F}_{p^2}$ operations is effective in speeding up the pairing since all operations can be decomposed into $\mathbb{F}_{p^2}$ operations

Because these curves have the embedding degree $k = 12$, the pairing computation involves performing the $\mathbb{F}_{p^{12}}$ arithmetic. We use the following extension:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[i]/(i^2 - \beta), \text{ where } \beta = -1, \quad (3)$$
$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^2}[w]/(w^6 - \xi), \text{ where } \xi = i + 1. \quad (4)$$

Since $\mathbb{F}_{p^{12}}$ is constructed on $\mathbb{F}_{p^2}$, it is important to accelerate the $\mathbb{F}_{p^2}$ operation in the optimal Ate pairing architecture.

Fig. 2 shows a top-down view of the operations required for a typical optimal Ate pairing computation. The optimal Ate pairing can be decomposed into two functions: f, called the Miller loop, and its power of $p^k - 1/r$, called the final exponentiation, where r is the order of the elliptic curve group. Each function is decomposed into finer-grained operations: elliptic curve operations, $\mathbb{F}_{p^{12}}$ operations, and $\mathbb{F}_{p^2}$ operations. The lowest-level operation is the prime field ($\mathbb{F}_p$) operation, i.e., addition and multiplication modulo p, which can be implemented with $\log p$-bit adders and modular multipliers. According to [22], the pairing computation for a 254-bit prime requires approximately 10,000 $\mathbb{F}_p$ multiplications and 57,000 $\mathbb{F}_p$ additions, indicating that high-throughput adders and modular multipliers are effective in speeding up the pairing computation. Because most operations for pairing can be decomposed into $\mathbb{F}_{p^2}$ operations, as shown in Fig. 2, we design a hardware architecture optimized for $\mathbb{F}_{p^2}$ operations by combining modular adders and multipliers.

B. BLS Signature

The BLS (Boneh–Lynn–Shacham) digital signature scheme [23], [1] is an example of PBC. The BLS digital signature has a short signature length than the conventional Elliptic Curve Digital Signature Algorithm (ECDSA) and a signature aggregation function, which can aggregate multiple signatures into a single signature. According to these advantages, some decentralized applications, such as DFINITY [24] and Ethereum [25], adopt the BLS signature to compress data and storage size.

$\mathbb{G}_1$ and $\mathbb{G}_2$ are elliptic curve groups, which have the same prime order p; g_1 and g_2 are the generators of $\mathbb{G}_1$ and $\mathbb{G}_2$, respectively, and H_0 is a hash function such that $H_0 : \mathcal{M} \to \mathbb{G}_1$. The BLS signature consists of three functions, KeyGen, Sign, and Verify.

- KeyGen(λ): For security parameter λ, it outputs secret key $sk \xleftarrow{\$} \mathbb{Z}_p$ and public key $pk \leftarrow [sk]g_2 \in \mathbb{G}_2$.
- Sign(sk, m): For an input message $m \in \mathcal{M}$, it outputs signature $\sigma \leftarrow [sk]H_0(m) \in \mathbb{G}_1$.
- Verify(pk, m, σ): Outputs "accept" if $e(H_0(m), pk) = e(\sigma, g2)$, otherwise "reject."

The Verify function determines whether the signature is acceptable using the pairing's property of biliniarlity.

In addition, the BLS signature allows the following Agg and AggVerify functions.

- Agg($\sigma_1, ..., \sigma_n$): For input n signatures, outputs $\mathcal{S} \leftarrow \sigma_1 + ... + \sigma_n$.
- AggVerify($(pk_1, m_1), ..., (pk_n, m_n), \mathcal{S}$): For n tuples of public keys and messages, outputs "accept" if $e(\mathcal{S}, g_2) = e(H_0(m_1), pk_1) \cdots e(H_0(m_n), pk_n)$, otherwise "reject."

Because the BLS signature can compress multiple signature data into one, owing to the Agg function, the BLS signature is expected to use to reduce network traffic and storage size.

C. Importance of Server-side FPGAs to Accelerate Pairing Computations

The main bottleneck of the BLS signature is $n + 1$ pairing computations involved in the AggVerify. As an example of the BLS signature application [26], [27], we take a wireless sensor network (WSN), where millions of sensor nodes sense some physical quantities and the sensed data is accumulated onto server nodes. Signing the sensed data on the sensor nodes provides the ability for authenticity verification on the server nodes. This prevents the data poisoning attack and the adversarial example attack. However, the signature length (64 bytes in typical ECDSA, for instance) is significantly larger than the sensed data (we estimate a few bytes in many cases). The load on the network bandwidth must be addressed when many nodes send signed data. Introducing the BLS signature onto the WSN significantly saves the network bandwidth by aggregating multiple signatures on intermediate nodes such as gateways. The server nodes must process $n + 1$ pairing computation to verify the aggregated signature, which can become a bottleneck.

Hardware implementation is a promising approach for accelerating pairing computation. Among hardware implementations, we focus on reconfigurable FPGA because security parameters can be updated. The server-side can use high-performance FPGAs, compared to IoT end devices. FPGAs have recently become available for cloud services with the Xilinx Virtex Ultrascale+ XVU9P. Our goal is, therefore, to implement a high throughput implementation on the Xilinx Virtex Ultrascale+ XVU9P. We propose a high-throughput implementation that can handle many pairing requests, focusing on the fact that the pairing requests on the server nodes occur in a batch-like manner when an aggregated signature arrives.

D. Related studies

Sakamoto et al. [15] proposed a pairing processor based on Yao's work [8], and their pairing processors have similar architectures. While Sakamoto et al. aimed to develop a low-latency and large-scale pairing processor with fully-unrolled Quotient Pipelining Montgomery Multiplication (QPMM), Yao et al. aimed to achieve a good area-speed efficiency with the Residue Number System (RNS). These studies have implemented pairing processors with 16 to 18 pipeline stages on FPGAs, and they demonstrated that their pairing processor can use the pipeline at high efficiencies of approximately 100%. This indicates that the pairing computation has fewer computation dependencies, and we can maintain the performance if we implement a deeper pipeline. We believe that we can improve the throughput of a pairing processor while maintaining the latency even when the pipeline stages are deepened to maximize the DSP operating frequency.

Acceraleting modular multiplication is a major topic in cryptography research, as modular multiplication is the dominant operation in computing many public key cryptosystems. Barrett reduction [28] and Montgomery reduction [29] are the most well-known modular multiplication algorithms, with the latter being the mainstream method.

The key point to implementing modular multiplication in FPGAs is effectively using DSPs, the dedicated hardware elements for performing multiplications. The operating frequency of the DSP is crucial in implementing a fast modular multiplier. Some studies [16], [17] aimed to operate the DSPs at the maximum frequency written in datasheets.

Suzuki [16] implemented a modular multiplier that operates at a maximum operating frequency of 400 MHz on Virtex-4 devices by adequately assigning the QPMM [30] algorithm to 17 DSPs. Gallin and Tisserand [17] implemented a 128-bit modular multiplier operating at 349 MHz, compared to the maximum operating frequency of 390 MHz for the Spartan-6 device, by adequately assigning the coarsely integrated operand scanning (CIOS)-Montgomery multiplication to nine DSPs. Although these studies proposed methodologies for maximizing the DSP operating frequency, their results were validated only on small-scale (older generation) FPGAs. We should take advantage of many resources (thousands of DSPs) in modern large-scale FPGAs used in cloud services.

Gallin and Tisserand [17] implemented an elliptic curve processor using their proposed modular multiplier. The elliptic curve processor requires adders and memories, and not only a modular multiplier. The adders and memories must also be designed appropriately to maintain a high operating frequency. In their study, the operating frequency of the elliptic curve processor was decreased from 349 MHz to 298 MHz.

III. Proposed Pairing Processor Architecture

A. Architecture overview

Fig. 3 shows the overall architecture of our pairing processor, which is an improved version of the one from our

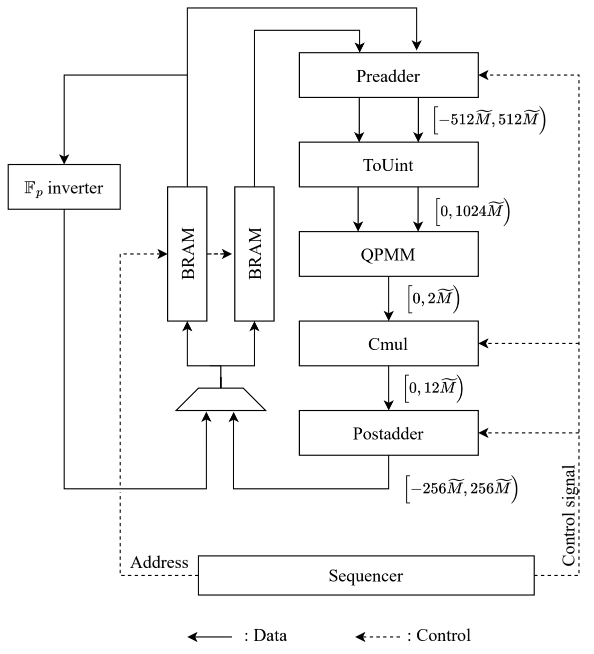


Fig. 3. Abstract overall architecture of the proposed pairing processor. The brackets $[,)$ represent the range of the output values of each module, where $\widetilde{M}$ is a constant for the modular multiplier algorithm. This architecture adopts the lazy-reduction technique for fast modular additions; hence, modules other than the QPMM extend the output ranges. The QPMM module performs all reduction operations, in which a value up to $1024\widetilde{M} - 1$ is reduced to up to $2\widetilde{M} - 1$

previous study [15]. This architecture is mainly designed to accelerate the $\mathbb{F}_{p^2}$ arithmetic frequently appearing in the pairing computation. For example, we can efficiently calculate the $\mathbb{F}_{p^2}$ multiplication and squaring using a Karatsuba-like formula as the following:

$$\begin{aligned} z_0 + z_1 i &= (x_0 + x_1 i) \cdot (y_0 + y_1 i) \\ &= x_0 y_0 - x_1 y_1 \\ &\quad + ((x_0 + x_1) \cdot (y_0 + y_1) - x_0 y_0 - x_1 y_1) i, \end{aligned} \tag{5}$$

$$\begin{aligned} z_0 + z_1 i &= (x_0 + x_1 i)^2 \\ &= (x_0 + x_1) \cdot (x_0 - x_1) + 2 x_0 x_1 i, \end{aligned} \tag{6}$$

where $z_0, z_1, x_0, x_1, y_0, y_1 \in \mathbb{F}_p$. Fig. 4 shows the ways in which the above equations are processed in the proposed architecture. As shown in the figure, the architecture enables the calculation of $\mathbb{F}_{p^2}$ multiplication or squaring by 3 or 2 cycles on average, respectively. In the following section, we explain ways in which this architecture achieves both high performance and a low SDR.

The pairing computation is performed on a finite field; therefore, all additions in the architecture require modular adders rather than simple adders. Because the modular arithmetic usually involves a comparison operation, the addition results must be determined before the comparison. This implies that we cannot take advantage of a fast redundant adder such as a CSA, and it is difficult to increase the operation frequency owing to the long carry chain. To solve this problem, we adopted the lazy-reduction technique into the architecture,

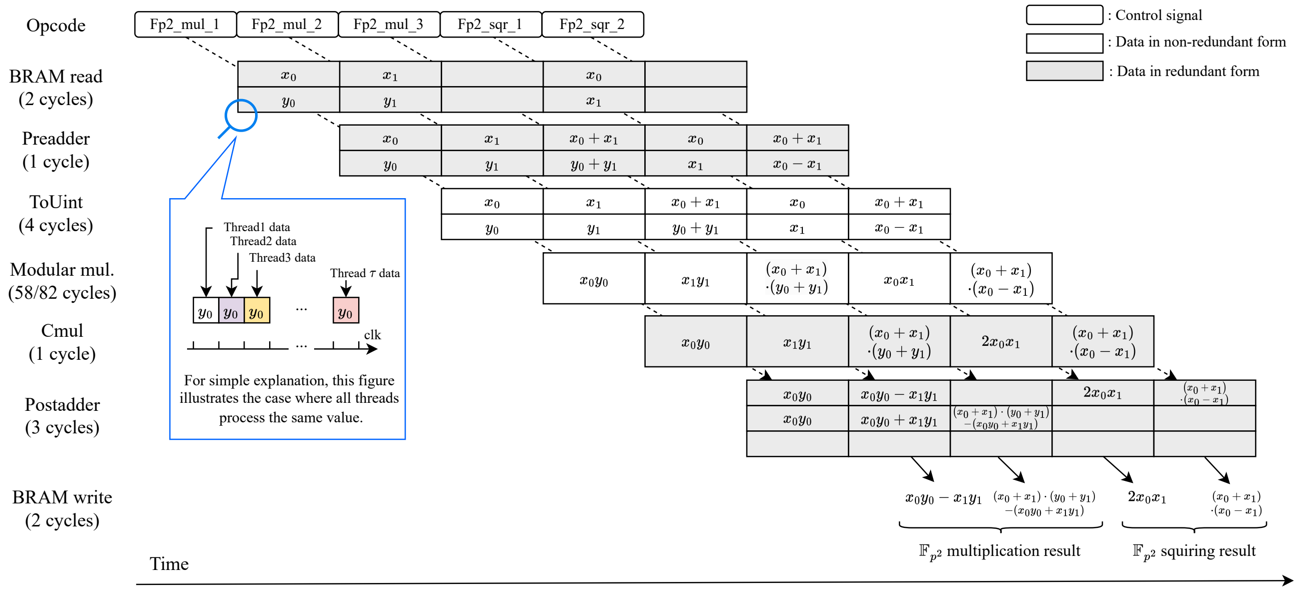


Fig. 4. Proposed architecture's pipeline data flow when processing $\mathbb{F}_{p^2}$ through multiplication and squaring for three and two cycles on average, respectively. The sequencer module first issues three opcodes (control signals) for the $\mathbb{F}_{p^2}$ multiplication, followed by two opcodes for $\mathbb{F}_{p^2}$ squaring. This figure shows the computation process according to the control signals. In the figure, all computations are in the same-sized boxes, but the computation time is different. This figure is not cycle-accurate, and the interleaved execution is omitted to simplify the explanation; τ data are read from the block random access memorys (BRAMs) for an issued opcode when enabling an interleaved execution. (We can also say that this figure shows the situation in which all threads are processing the same data.)

which separates modulo operations from the modular adders and pushes all the modulo operations to the modular multiplier. This enables all adders in the architecture to be converted to redundant adders, thus increasing the operating frequency.

Although CSA is the most popular redundant adder that can eliminate a long carry chain, it has a disadvantage of doubling the number of used FFs (Flip Flops). Consequently, for Xilinx FPGAs that have dedicated logic for carry propagation (CARRY8), a ripple-carry adder (RCA) is often more efficient. To avoid a full carry chain of RCA, we employ the partially redundant partitioned adder (PRPA) that divides an adder into κ sub-adders and saves an γ-bit carry for each sub-adder. For a non-redundant, R-bit integer A represented as

$$A = \sum_{i=0}^{\kappa-1} a_i \cdot 2^{r \cdot i} \quad (a_i \in [0, 2^r - 1]), \tag{7}$$

we define a partially redundant κ-partitioned integer $\hat{A}$ as follows:

$$\hat{A} = \sum_{i=0}^{\kappa-1} \hat{a}_i \cdot 2^{r \cdot i} \quad (\hat{a}_i \in [0, 2^{r+\gamma} - 1]), \tag{8}$$

where $r = R/\kappa$ (we assume $\kappa | R$). In the PRPA, each sub-addition is performed with $r + \gamma$-bit width to eliminate the original R-bit carry chain. The length of the each carry bit depends on the number of additions from the output of the modular multiplier to the next input ($\gamma = 8$ bits are sufficient for our scheduling). These saved carry bits are used by the ToUint module in multiple cycles to maintain a high operation frequency. We implemented a 256-bit two-input RCA on an xcvu9p-2l device and found that it operates at approximately 400 MHz. Because the BLS12_381 pairing requires a 381-bit addition, we selected $\kappa = 4$ with an operating margin.

B. Operation Modules

1) Sequencer: The sequencer issues control signals to control the operation of each module. The sequencer has a built-in ROM with 2048 entries, which contains a pairing computation program optimized for the 18-stage pipeline architecture [15]. Because the pipeline of our architecture has more than 18 stages (89 stages for BN254 pairing and 121 stages for BLS12_381 pairing), this program cannot run efficiently. We make our architecture to support an interleaved execution (multi-threading), where a core can concurrently execute multiple pairings. For the number of threads τ, the sequencer issues the control signal every τ cycles and issues the memory address for corresponding threads every cycle.

Let σ be the number of pipeline stages of the entire pairing processor. Our program works under the condition that $\sigma < 18\tau$; therefore, we set $\tau = 5$ for the BN254 pairing and $\tau = 7$ for the BLS12_381 pairing. Our pairing program is optimized by various techniques described in [3], including twisted elliptic curves, sparse multiplication, and compressed squaring techniques (see [15] for the detailed scheduling).

2) Preadder: The preadder shown in Fig. 5 is a two-input two-output module for generating the input to the subsequent modular multiplier. The data path is divided into two parts: the left and right sides produce the multiplicand and multiplier of the modular multiplier, respectively. The main arithmetic units of the preadder are four adders (adder1-4 constructed as the PRPAs), where adder3 is a subtractor taking two's complement value as the right-side input. This module has three operational

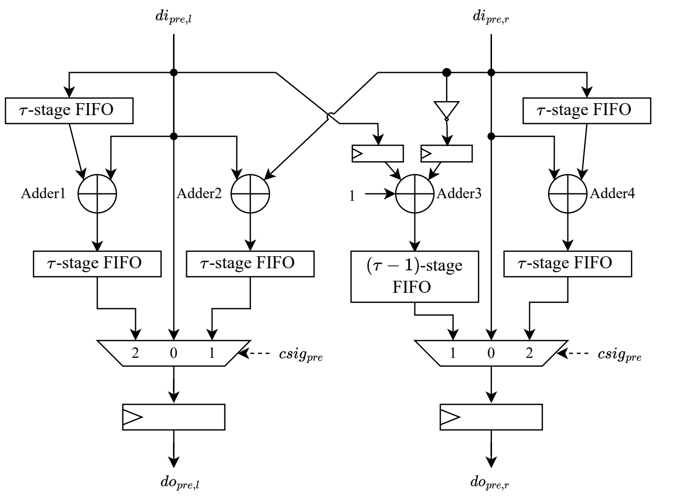
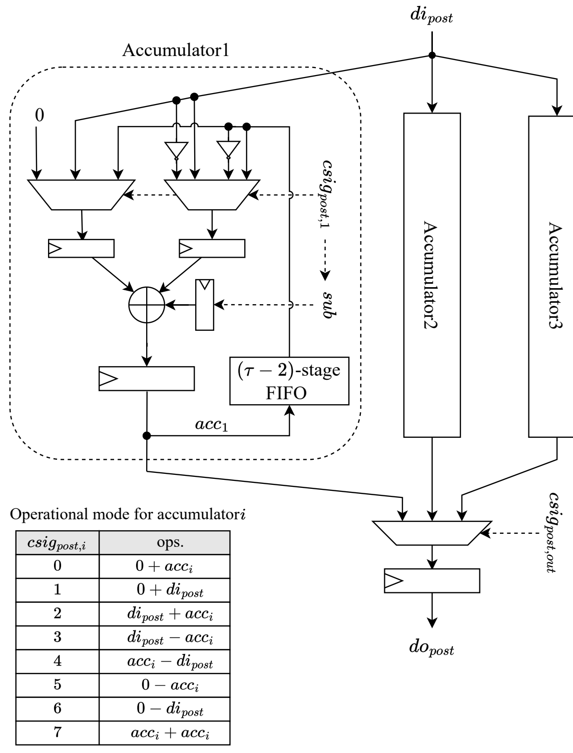


Fig. 5. Preadder block diagram. A preadder has three operational modes controlled by the control signal $csig_{pre}$: (1) pass-through mode that directly outputs di when $csig_{pre} = 0$, (2) one-step-delay mode that outputs the sum of $di_{pre,l}$ and $di_{pre,r}$ of τ-cycles-before when $csig_{pre} = 1$, and (3) two-step-delay mode that outputs the sum of $di_{pre,l}$ and $di_{pre,r}$ of 2τ-cycles-before when $csig_{pre} = 2$.



$csig_{post,i}$	ops.
0	$0 + acc_i$
1	$0 + di_{post}$
2	$di_{post} + acc_i$
3	$di_{post} - acc_i$
4	$acc_i - di_{post}$
5	$0 - acc_i$
6	$0 - di_{post}$
7	$acc_i + acc_i$

Fig. 6. Postadder block diagram. This module has three accumulators; each consists of the PRPA, two selectors, and a register. The operational mode of an accumulator depends on the selector outputs controlled by the control signal $csig_{post,1}$. When a sub signal, which is a bit of $csig_{post,1}$ equals one, the right-hand selector selects one of the inverted inputs, realizing a subtraction operation using two's complement value. The postadder output is selected from three accumulators by the control signal $csig_{post,out}$.

modes as shown in the figure. First-in first-out FIFO registers are for supporting τ interleaved execution.

3) ToUint module: The ToUint module is a two-input two-output module responsible for converting the PRPA form values to non-redundant form and making the values unsigned. We realize the unsigned operation by adding $512\tilde{M}$ to the input values because the minimum input value is $-512\tilde{M}$ by carrying up the redundant γ bits, taking $\kappa = 4$ cycles to eliminate the long carry propagation.

4) Cmul: The Cmul module is a two-input two-output module responsible for multiplying by constants $\alpha \in \{2, 3, 4, 6\}$, which consist of constant shifters and the PRPA. Multiplication by two is useful for accelerating $\mathbb{F}_{p^2}$ squaring which, has a term of $2x_0x_1$ (see equation (6)). Multiplications by 3, 4, and 6 are useful for accelerating $\mathbb{F}_{p^{12}}$ and elliptic curve operations.

5) Postadder: The postadder module shown in Fig. 6 is a one-input one-output module responsible for accumulating the output of the QPMM. The postadder consists of three accumulators, two of them are mainly used for producing the two terms of an $\mathbb{F}_{p^2}$ element, as shown in Fig. 4, and the remainder is used for calculating $\mathbb{F}_{p^{12}}$ elements. Each accumulator has eight modes of operations controlled as shown in the figure. Similar to the preadder, the postadder has FIFO registers to support the interleaved execution.

6) $\mathbb{F}_p$ inverter: In our pairing implementation, an $\mathbb{F}_p$ inversion occurs only once per pairing; however, the inverse operation can be a bottleneck if executed on the QPMM by Fermat's little theorem, because the computation causes many pipeline bubbles. To solve this bottleneck, our pairing processor has a dedicated module for $\mathbb{F}_p$ inversion, which implements the Montgomery inverse algorithm [31] with τ-stage pipeline.

7) QPMM: The improvement of the modular multiplier module is the main part of our proposed method. The detailed description is presented in the next subsection III-C. So far, we discussed that our architecture consumes many slices for adders to accelerate $\mathbb{F}_{p^2}$ operations and FFs to support the interleaved execution. Therefore, to maximize the throughput in multi-core implementations, the modular multiplier must be implemented with a small SDR by efficiently utilizing the functions of DSP primitives.

C. Low-SDR High-Throughput Modular Multiplication Algorithm

The basic idea of lowering the SDR is to improve performance while consuming a large amount of DSP with loop-unrolled implementation. The following section first describes the features of the Xilinx DSP48E2 primitive and proposes a modular multiplication algorithm suitable for using its asymmetric multiplier and sum-of-products arithmetic functions.

1) DSP48E2: Many FPGAs contain DSP hardware macros to accelerate signal-processing applications. The specifications of the DSP vary by vendor and device. Virtex Ultrascale+ devices are the mainstream FPGAs available on many cloud services, AWS F1, Huawei, and Alibaba cloud. This study proposes a modular multiplication algorithm for DSP48E2, which is the DSP primitive on Virtex Ultrascale+ FPGAs.

A simplified block diagram of DSP48E2 is shown in Fig. 7. DSP48E2 takes 27-bit A, 18-bit B, 48-bit C, and PCIN as

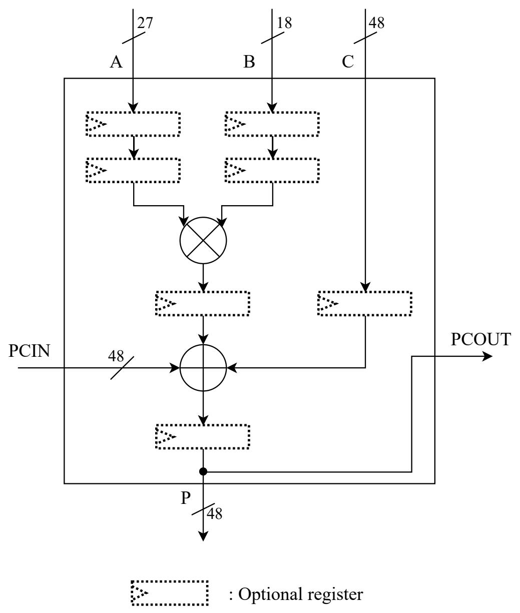


Fig. 7. Simplified block diagram of the DSP48E2 primitive.

inputs and outputs 48-bit P and PCOUT. PCIN is a dedicated line connected to PCOUT of the adjacent DSP, and it is used to input the results of the adjacent DSP with a minimum delay. DSP48E2 is a typical MAC (multiply-accumulate) unit, with a 27×18-bit signed multiplier (26×17-bit unsigned multiplier) and a 48-bit three-input adder followed by the multiplier. DSP48E2 has optional pipeline registers at various locations, which can be used to increase the operating frequency instead of the latency. The maximum operating frequency of the DSP is listed in the datasheet [32], for example, up to 644 MHz for Virtex Ultrascale+ xcvu9p-2l devices that are available on the cloud services.

2) High-Throughput Modular Multiplication Algorithm for DSP48E2: We propose a modified QPMM algorithm [30, Algorithm 4] suitable for DSP48E2. The naive QPMM algorithm is shown in Algorithm 1. QPMM is a variant of Montgomery multiplication that eliminates computational dependency in the conventional high-radix Montgomery multiplication algorithm [30, Algorithm 1]. The essential improvement of QPMM is the ability to delay the timing when q_i is required by d cycles. This eliminates the computational dependency between L_1 and L_2 and allows them to be executed in parallel. A larger value of d makes the multiplication width long; therefore, d should be minimal. Moreover, because q_i is the lower k bits of S_i, carry bits must not be propagated to more than k bits during L_2 calculation. Thus, we can efficiently calculate most additions during the L_2 calculation using redundant representation adders such as the CSA.

To execute the QPMM algorithm on the DSP48E2, we must replace the L_2 calculation with a multiple-precision operation matched to the DSP48E2 operation width. Because the multiplier of DSP48E2 is asymmetric, we represent multiplicand A and multiplier B using the different radixes 2^ℓ and 2^k (typically $\ell = 26, k = 17$), respectively, as the following:

Algorithm 1 Quotient Pipelining Montgomery Multiplication (QPMM) [30]

```
Require: radix: $2^k$, delay: $d$, modulus: $M > 2$, $\gcd(M,2) = 1$, $(-MM') \bmod 2^k = 1$, $\tilde{M} = (M' \bmod 2^{k(d+1)})M$, $\widetilde{M} = (\tilde{M}+1) \operatorname{div} 2^{k(d+1)}$, $4\tilde{M} < 2^{kn} = R$, $0 \le A, B \le 2\tilde{M}$, $B = \sum_{i=0}^{n+d}(2^k)^i b_i$, $b_i \in \{0,1,\cdots,2^k-1\}$, $b_i = 0$ for $i \ge n$.
Ensure: $S_{n+d+2} \equiv ABR^{-1} \pmod M$, $0 \le S_n \le 2\tilde{M}$.
1: $S_0 := 0$; $q_{-d} := 0; \cdots; q_{-1} := 0$
2: for $i := 0$ to $n+d$ do
3:     $L_1 : q_i \quad := S_i \bmod 2^k$
4:     $L_2 : S_{i+1} := S_i/2^k + q_{i-d}\widetilde{M} + b_i A$
5: end for
6: $L_3 : S_{n+d+2} := 2^{kd} S_{n+d+1} + \sum_{j=0}^{d-1} q_{n+j+1} 2^{kj}$
7: return $S_{n+d+2}$;
```

Algorithm 2 $d = 0$ QPMM for DSP48E2.

```
Require: $0 \le A \le 2\tilde{M}, 0 \le B \le 2\tilde{M}, k < \ell, k < 17, \ell < 26$, $4\tilde{M} < 2^{kn} = R$, $c = \ell - k$, $4\tilde{M} < 2^{\ell m}$, $\widetilde{m_j}$: The $j$-th digit of radix-$2^\ell$ represented $\widetilde{M}$ for $0 \le j < m$ and $\widetilde{m_{m-1}} = 0$, other parameters are the same as Algorithm 1.
Ensure: $S_{n+2} \equiv ABR^{-1} \pmod M$ and $0 \le S_{n+2} < 2\tilde{M}$.
1: $s_{0,0} \cdots s_{0,m+1} := 0$
2: for $i = 0$ to $n$ do
3:     $s_{i+1,0} := sl_{i,1} + s_{i,0}/2^k$
4:     $q_i := s_{i+1,0} \bmod 2^k$
5:     for $j = 0$ to $m-1$ do
6:         $s_{i+1,j+1} := a_j b_i + q_i \widetilde{m_j} + sl_{i,j+2} 2^c + su_{i,j+1} 2^k$
7:     end for
8: end for
9: $S_{n+2} := s_{n+1,0}/2^k + \sum_{j=1}^{m} 2^{\ell(j-1)} s_{n+1,j}$
10: return $S_{n+d+2}$;
```

$$A = \sum_{j=0}^{m-1} (2^\ell)^j a_j, \quad B = \sum_{i=0}^{n} (2^k)^i b_i. \tag{9}$$

Consider all values except B in the 2^ℓ-radix representation. The sum of $q_{i-d}\widetilde{M}$ and $b_i A$ becomes the sum of products of the same index j as follows:

$$q_{i-d}\widetilde{M} + b_i A = \sum_{j=0}^{m-1} (\widetilde{m_j} q_{i-d} + a_j b_i) \cdot 2^{j\ell}. \tag{10}$$

For $c = \ell - k \ (k \le \ell)$,

$$S_i/2^k = s_{i,0}/2^k + \sum_{j=1}^{m-1} s_{i,j} \cdot 2^c \cdot 2^{\ell(j-1)}, \tag{11}$$

and let $t_{i,j} = \widetilde{m_j} q_{i-d} + a_j b_i$, then the sum of equation (10) and (11) is

$$L_2 : S_{i+1} = s_{i,0}/2^k + \sum_{j=1}^{m-1} (s_{i,j} \cdot 2^c + t_{i,j-1}) \cdot 2^{\ell(j-1)}. \tag{12}$$

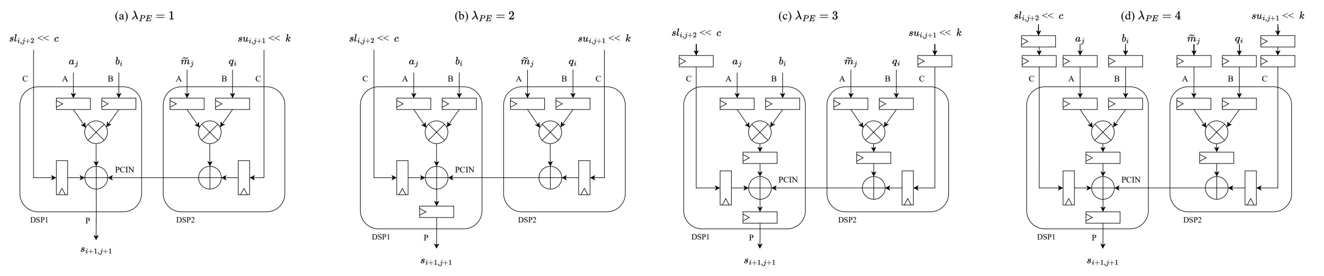


Fig. 8. Processing element (PE) designs for four types of latency λ_{PE}. (a) $\lambda_{PE} = 1$ PE design. (b) $\lambda_{PE} = 2$ PE design. (c) $\lambda_{PE} = 3$ PE design. (d) $\lambda_{PE} = 4$ PE design.

DSP48E2 can calculate equation (12) easily; however, the length of $s_{i,j}$ increases by c bits as index i progresses. Let the lower $2k$ bits of $s_{i,j}$ be $sl_{i,j} = s_{i,j} \bmod 2^{2k}$ and the remaining upper bits be $su_{i,j} = s_{i,j}/2^{2k}$. Equation (11) is rewritten as follows:

$$\begin{aligned} S_i/2^k &= s_{i,0}/2^k + \sum_{j=1}^{m-1} (sl_{i,j} + su_{i,j} \cdot 2^{2k}) \cdot 2^c \cdot 2^{\ell(j-1)} \\ &= s_{i,0}/2^k + \sum_{j=1}^{m-1} (sl_{i,j} + su_{i,j}) \cdot 2^{2k+c+\ell(j-1)}. \end{aligned} \tag{13}$$

Because $2k + c + \ell(j-1) = \ell - c + \ell j = k + \ell j$,

$$S_i/2^k = sl_{i,0}/2^k + \sum_{j=1}^{m-1} (sl_{i,j} \cdot 2^c + su_{i,j-1} \cdot 2^k) \cdot 2^{\ell(j-1)}, \tag{14}$$

$$\begin{aligned} L_2 : S_{i+1} &= sl_{i,0}/2^k \\ &+ \sum_{j=1}^{m-1} (sl_{i,j} \cdot 2^c + su_{i,j-1} \cdot 2^k + t_{i,j-1}) \cdot 2^{\ell(j-1)}, \end{aligned} \tag{15}$$

where each $s_{i,j}$ is at most $k + \ell + 1$ bits or less. When $\ell = 26, k = 17$, $k+\ell+1 = 44$. DSP48E2 can output up to 48 bits; therefore, we can efficiently calculate these sum-of-products operation within the DSP.

We propose a QPMM variant algorithm using Equation (14), as shown in Algorithm 2. Assigning the four-term sum-of-products at Line 6 to two DSPs, as shown in Fig. 8, we can complete most QPMM operations within the DSPs. We refer to the combination of these two DSPs as a processing element (PE). The figure shows four different latency PEs using optional registers in the DSPs. The overall image of the unrolled implementation of Algorithm 2 using these PEs is shown in Fig 9. The demultiplexers (DeMUX) virtually have no cost because it is a distribution of wires, and most functions except for few adders, can be completed within the DSP, which is expected to contribute to a low SDR.

The algorithm parameters are $k = 17, \ell = 26, n = 17, m = 11$ for BN254, and $k = 17, \ell = 26, n = 25, m = 16$ for BLS12_381, selected for extending R to perform the lazy reduction ($R = 2^{289}$ for BN254 and $R = 2^{425}$ for BLS12_381). Ma et al. [33] reported that increasing 2 bits in R can extend the range of QPMM input A and B twice. These parameters extend the maximum input of Algorithm 2 from $2\tilde{M}$ to $1024\tilde{M}$.

IV. Evaluation results

A. Evaluation platform

We implemented the proposed method on a VCU118 evaluation board and evaluate area and timing performances after Place and Route (PAR) by Vivado 2020.1. We use *default* and *performance explore* synthesis strategies for the modular multiplier and the pairing core evaluation, respectively. The VCU118 board contained a 16 nm Virtex UltraScale+ xcvu9p-2l FPGA, which has 147,780 CLBs (1,182,240 6-input LUTs and 2,364,480 FFs), 6,840 DSPs, and 2,160 36kb BRAMs. The xcvu9p is a high-performance large-scale FPGA for use in server-side applications and is adopted in Alibaba, Huawei, and AWS clouds. Because the notations of the area differ between Ultrascale and other devices, we converted certain values in the implemented results, such that 1 RAMB36 = 2 RAMB18s and 1 CLB = 2 slices.

B. Modular multiplier evaluation

We implemented Algorithm 2 with a fully-unrolled manner on the VCU118 board. The performance evaluation results for BN254 and BLS12_381 (254-bit and 381-bit multipliers) on each PE latency are shown in Table I. As the latency increased, the operating frequency improved, and when $\lambda_{PE} = 4$, the 256-bit modular multiplier operated at 623 MHz, which is close to the maximum operating frequency of 644 MHz [32]. The latency and the resource consumption worsened as λ_{PE} was increased; however, we do not consider these to be problems as our goal is to maximise throughput. Even at $\lambda_{PE} = 4$, the SDR remains below 43, around 20, indicating that the DSP is being used effectively. For these results, we adopted a $\lambda_{PE} = 4$ design that has the best throughput, for the pairing processor evaluation in the next subsections.

Table II shows the comparisons of our proposed multipliers ($\lambda_{PE} = 4$) to those from previous studies. The proposed method achieved the highest throughput with the largest Slice and DSP resource consumption compared, to the existing methods. The latency of the proposed method is also comparable to other methods. The proposed method also has the best TP/ESlices, a measure of area efficiency, indicating that it is suitable for server-side FPGAs, where many resources are available. Furthermore, the proposed method achieved the lowest SDR. This indicates that the slice consumption does not become a bottleneck when the proposed multiplier is

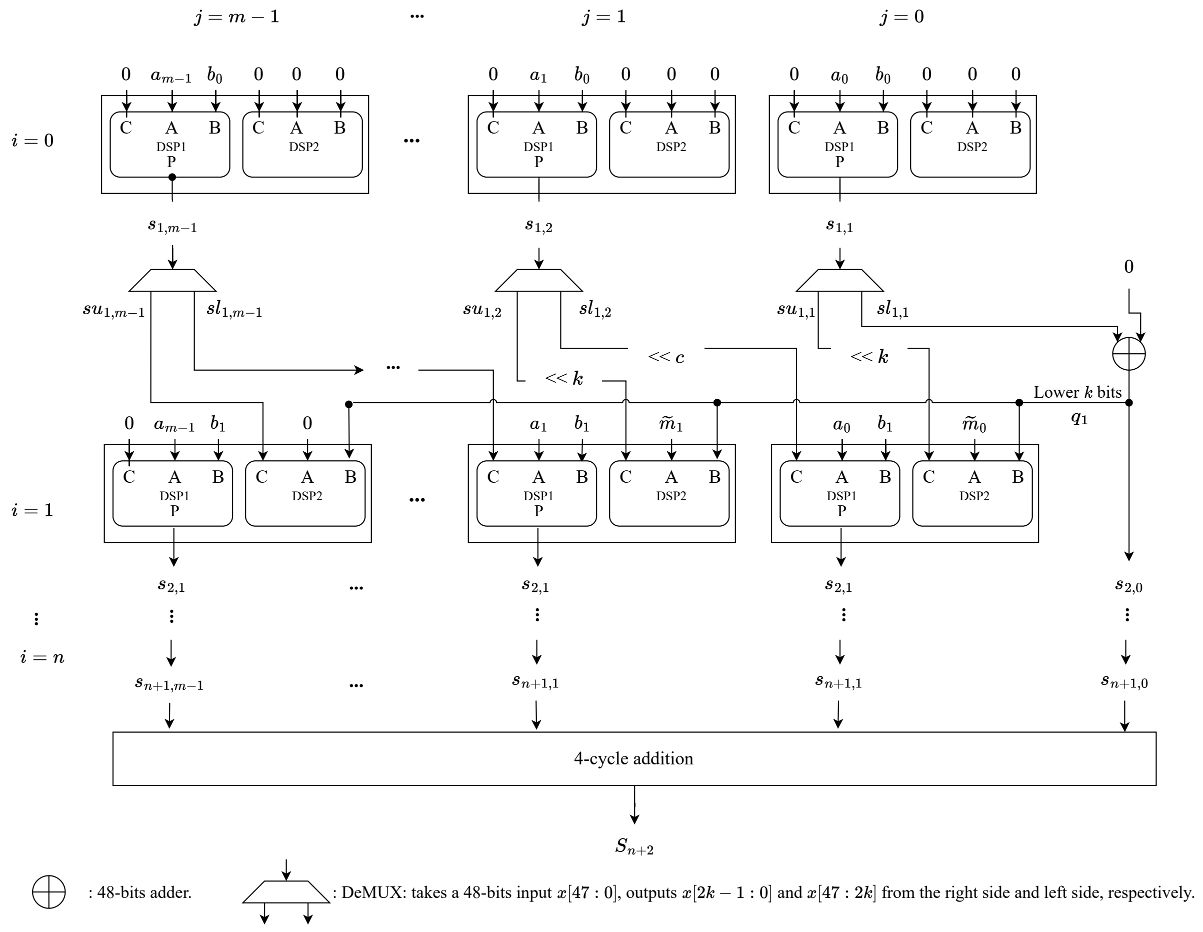


Fig. 9. Fully-unrolled QPMM design (Algorithm 2) using PEs. DeMUX elements can be implemented as wire splitting with no cost. Hence, although the right-most adders and the final four-cycle addition should be implemented with few slice resources, almost the entire QPMM circuit can be implemented with DSP resources. This is the reason this module achieves a low SDR and high performance. This module can exhibit the DSP‘s maximum frequency without making the slices a bottleneck.

embedded in a pairing processor that spends many slices for accelerating modular adders.

The papers [34] and [35] evaluate modular multipliers on the same generation devices as ours. [35] is a low-latency architecture that uses a large number of DSPs and LUTs to complete the modular multiplication in a single cycle, achieving a latency of about 15% of our implementation. Meanwhile, the throughput is less than a tenth of our implementation, making it unsuitable for server-side applications. To compare our implementation with [34], we need to take into account the resource utilization; however, we cannot use the SDR and ESlice metrics of resource utilization because [34] does not provide the number of slices information. If [34] were implemented with 62 cores in parallel, the throughput and resource utilization would be approximately 62 times larger, and thus Throughput/LUTs/FFs/DSPs = 78.05/39.618/81.158/372, which is comparable to our implementation and number of DSPs. In this case, our implementation achieves 1.5 times the throughput with about $1/12$ and $1/3$ of the LUTs and FFs.

The comparisons in the table do not consider the difference of device generations. The proposed method has been evaluated on the latest FPGA, Virtex Ultrascale+, whereas most existing methods have been evaluated on Virtex-7, which is two generations older. The manufacturing process has been shrunk from 28 nm to 14 nm between these generations; therefore, we should consider the increase in the operating frequency. Considering the maximum operating frequency of the DSPs, Ultrascale$+$ is 644 MHz while Virtex-7 is 650 MHz, and there is no significant difference. If the operating frequency from a previous study running near the DSP's maximum frequency was evaluated on VCU118 [17], it would be bounced to the maximum frequency and almost have a similar performance. Conversely, it is difficult to predict the increase in the performance of implementations that are not near the maximum frequency limit. As an example, we estimate the frequency difference in one device generation to be at most 1.22 times, based on the results [36], which are from evaluations of the same multipliers on different FPGAs, Virtex-6 and Virtex-7. Even when assuming a frequency difference of two generations ($1.48\times$), our implementation achieves the best TP/ESlices.

TABLE I
PERFORMANCE EVALUATION RESULTS OF OUR PROPOSED 254-BIT AND 381-BIT MODULAR MULTIPLIERS IMPLEMENTED ON VCU118. λ_{PE} AND λ_{mul} REPRESENT THE LATENCIES OF THE DSP AND THE ENTIRE MODULAR MULTIPLICATION, RESPECTIVELY.

Size	λ_{PE}	λ_{mul}	Slices / LUTs / FFs / DSPs	ESlices	SDR	Freq. [MHz]	Latency [ns]	Throughput [Gbps]	TP/ESliecs $\times 10^{-3}$
254	1	22	3,142 / 733 / 6,815 / 367	18,923	8.56	251	87.42	63.92	3,377
254	2	40	3,566 / 801 / 11,933 / 367	19,347	9.71	431	92.70	109.59	5,664
254	3	58	5,970 / 3,096 /14,978 / 367	21,751	16.26	600	96.66	152.40	6,749
254	4	76	7,398 / 3,262 / 28,901 / 367	23,179	20.16	623	121.87	158.38	6,832
381	1	30	7,044 / 1,262 / 13,550 / 790	41,014	8.91	217	138.24	82.67	2,015
381	2	56	8,404 / 1,331 / 23,582 / 790	42,374	10.63	362	154.69	137.92	3,254
381	3	82	9,932 / 6,415 / 28,508 / 790	43,902	12.57	455	180.21	173.36	3,948
381	4	108	17,286 / 6,796 / 61,248 / 790	51,256	21.88	496	217.74	188.98	3,686

ESlices: Equivalent slices = Slices + 43 × DSPs, where 43 is the ratio of slices to DSPs on entire FPGA resources.
TP/ESlices: Throughput per ESlices = $\frac{\text{Throughput}}{\text{ESlices}}$.

TABLE II
PERFORMANCE COMPARISONS OF THE PROPOSED MODULAR MULTIPLIERS AND THOSE FROM PREVIOUS STUDIES.

Design	Size	Device	Slices / LUTs / FFs / DSPs	SDR	ESlices	Freq. [MHz]	Latency [ns]	Throughput [Gbps]	TP/ESlices $\times 10^{-3}$
Ours	254	xcvu9p	7,398 / 3,262 / 28,901 / 367	20.16	23,179	623	121.87	158.38	6,873
Ours	381	xcvu9p	17,286 / 6,796 / 61,248 / 790	21.88	51,256	496	217.74	188.98	3,686
[35]	256	xcvu9p	NA / 87,622 / 2,952 / 289	NA	NA	57	17.55	14.58	NA
[34]	256	xczu9eg	NA / 639 / 1,309 / 6	NA	NA	625	203	1.26	NA
[17]	256	Virtex-7	314 / 634 / 778 / 9	34.88	701	598	239.13	2.28	3,264
[37]	192	xc5vlx330	500 / NA / NA / 12	41.67	1016	240	91.67	2.09	2,061
[38]	255	xc7z020	1354 / 2779 / 3958 / 40	33.85	3074	305.15	206.45	4.94	1,607
[39]	256	Virtex-7	937 / 2,480 / 3565 / 40	23.42	2657	207.1	222.11	3.45	1,301
[40]	434	xc7vx690	NA / 1317 / 2677 / 46	NA	1978	357	249.29	2.53	NA
[36]	384	xc7vx485t	NA / 922 / NA / 0	NA	NA	155.2	2480	0.154	NA
[36]	384	xc6vsx475t	NA / 923 / NA / 0	NA	NA	127.3	3030	0.126	NA

ESlices = Slices + 43 × DSPs, where 43 is the ratio of slices to DSPs on entire FPGA resources.
TP/ESlices: Throughput per area = $\frac{\text{Throughput}}{\text{ESliecs}}$.

C. Pairing processor evaluation

1) Single-core evaluation: Table III shows the performance comparison of the single-core pairing implementation. In this table, our pairing processors are evaluated using $\lambda_{PE} = 4$ because this parameter has the best throughput and the closest SDR to 43. Note that we can adjust the trade-off between latency and throughput of the pairing computation using the different design modular multipliers with $\lambda_{PE} = \{1, 2, 3\}$.

Our implementation executes $\tau = 7$ pairings concurrently and completes them with 452 µs. As shown in the table, Opasatian and Ikeda's implementation [13] shows the best latency and throughput performance. They implemented a low latency core using a look-up-table modular reduction technique and evaluated it on a same generation FPGA as ours. The throughput of their implementation exceeds our implementation by approximately four percent. However, their implementation uses a large number of LUTs to perform modular reduction, which means that it has the SDR more than 43, expecting to make worse the performance of multi-core implementation. Since their paper shows only LUT utilization (225,607), we estimated their slice utilization as $56,401 = 225,607/4$. Using this value, the SDR of their implementation is calculated as 87.03, which is more than twice of our implementation‘s, indicating that the number of slices restricts the number of cores under multi-core implementation, and lots of DSPs cannot be used. On the other hand, our implementation has a good SDR (41.24) and are expected to perform better performance under multi-core evaluation. Actual multi-core performance comparisons are given in Section IV-C2.

Devlin [11] implemented a large-scale, server-side pairing processor and evaluated their implementation on xcvu37p, which is the same generation device to our xcvu9p. These devices have almost the same performance and circuit resources; thus, their work is most comparable to our implementation results. Our implementation achieved approximately 9.8 times throughput and 0.71 times latency, compared to Devlin's implementation. Although our implementation uses more than twice as much DSPs as Devlin's, the slice usage is less than 1/2; therefore, their implementation had a slice bottleneck and did not take advantage of the fast DSP primitives. Furthermore, our implementation has SDR = 41.24, whereas theirs has SDR = 236.95, meaning the performance difference will be spread for multi-core implementations. Compared to the state of the Art software implementation [42], our implementation shows the better performance of approximately 10 times throughput and 0.7 times latency.

For the BN254 curve, our implementation runs at 590 MHz, which is close to 644 MHz of the DSP's maximum frequency, and completes $\tau = 5$ pairings in 187 µs. Compared to the previous studies, our pairing implementation achieved the best throughput, comparable latency, and the best area efficiency (TP/ESlices). Previous work [15] is the only one that shows the evaluation results on the same device. It aims for a low-latency pairing implementation and achieves the best latency of 102 µs. The proposed method improves throughput by 2.7 times at the cost of 1.8 times worse latency. This seems like a good compromise. Note that our implementation allows the trade-

TABLE III
SINGLE-CORE PAIRING PERFORMANCE EVALUATION.

Design	Curve	Device	SG	Slices / DSPs / BRAMs	SDR	Freq. [MHz]	Latency [μs]	Throughput [pairings/s]	TP/ESlices $\times 10^3$
Ours	BLS12_381	xcvu9p	-2l	32,582 / 790 / 28.5	41.24	458	452	15,477	232.5
[13]	BLS12_381	xcvu13p	NA	56,401 / 648 / 13	87.03	180	62.2	16,077	190.7
[11]	BLS12_381	xcvu37p	-2	81,750*/ 345 / 231$^+$	236.95	200	633	1,580	16.35
[9]	BLS12_381	Kintex-7	NA	1,006 / 0 / NA	NA	200	36,140	27	26.83
[42]	BLS12_381	Core i7-7700	NA	NA / NA / NA	NA	3,600	650	1,538	NA
Ours	BN254	xcvu9p	-2l	17,230 / 367 / 20.5	46.94	590	187	26,602	805.8
Ours [15]	BN254	xcvu9p	-2l	10,842 / 460 / 14	23.56	173	102	9,803	320.1
[14]	BN254	xczu9eg	-2	3,746 / 36 / 18	104.05	230	952	1,050	198.3
[10]	BN254	xc6vlx240t	NA	5,570 / 30 / NA	185.66	225	350	2,857	416.4
[41]	BN254	xc6vlx240t	-3	5,163 / 144 / 21	35.80	166	375	2,666	234.4
[12]	BN254	xc7vx690	-2	2,506 / 40 / 20	62.65	180	14,140	70	16.56
[42]	BN254	Core i7-7700	NA	NA / NA / NA	NA	3,600	250	4,000	NA

* Estimated value, $^+$ Use 133 URAMs other than BRAM.
SG: Speed grade.
TP/ESlices: Throughput per ESlices $= \frac{\text{Throughput}}{\text{ESliecs}}$, where ESlices = Slices + 43 × DSPs.

TABLE IV
MULTI-CORE PERFORMANCE EVALUATION. VALUES IN ITALIC ARE OUR ROUGH (THE BEST-CASE ESTIMATION)ESTIMATION. THROUGHPUT IN THE BRACKET ARE THE STRICT (WORST-CASE) ESTIMATION.

Design	Curve	Device	SG	#Cores	Slices / DSPs / BRAMs	Freq. [MHz]	Latency [μs]	Throughput [pairings/s]
Ours	BLS12_381	xcvu9p	-2l	7	203,694 / 5,530 / 199.5	381	544	90,055
[13]	BLS12_381	xcvu13p	NA	*5*	*282,005 / 3,240 / 65*	*180*	*62.2*	*80,385 (16,077)*
[11]	BLS12_381	xcvu37p	-2	*3*	*245,250 / 1,035 / 693*	*200*	*633*	*4,740*
[9]	BLS12_381	Kintex-7	NA	*293*	*294,758 / 0 / NA*	*200*	*36,140*	*7,911*
Ours	BN254	xcvu9p	-2l	16	216,906 / 5872 / 328	393	281	284,127
Ours [15]	BN254	xcvu9p	-2l	*14*	*151,788 / 6,440 / 196*	*173*	*102*	*137,242*
[14]	BN254	xczu9eg	-2	78	292,188 / 2,808 / 1,404	230	952	81,900
[10]	BN254	xc6vlx240t	NA	*53*	*295,210 / 1,590 / NA*	*225*	*350*	*151,421*
[41]	BN254	xc6vlx240t	-3	*47*	*242,661 / 6,768 / 987*	*166*	*375*	*125,302*
[12]	BN254	xc7vx690	-2	*108*	*270,648 / 4,320 / 2,160*	*180*	*14,140*	*7,560*
[43]	BN254	RTX3060	NA	NA	NA/ NA / NA	1,320	NA	43,856

SG: Speed grade

off between latency and throughput to be tuned by changing the number of pipeline stages in the modular multiplier.

It is difficult to directly compare performance on different generation FPGAs (Virtex-6 or -7), but we can highlight some of the advantages of our architecture. First, our implementation have the SDR, which is a device-generation-independent metric, close to 43; this indicates a good balance of DSP and slice usage. In addition, because operating frequency is the main difference when the same architecture is implemented on different FPGAs, we can predict the performance on other FPGAs by considering the frequency difference between different FPGAs. In [36], a modular multiplier was evaluated on Virtex-6 and -7 devices, and the frequency difference was approximately 1.22 times. Assuming 1.22 times frequency difference in one generation, we can infer a difference of approximately 1.82 times in three generations. If the proposed architecture is implemented on a Virtex-6 FPGA, which is a three generation older device than ultrascale+, the frequency can be estimated to be 324 MHz. The throughput would be 14,616 pairings/s. Even in this case, the throughput of the proposed architecture is the largest among existing studies.

2) Multi-core evaluation: This section compares multi-core performance for the pairing on the target FPGA xcvu9p (see table IV). While the performance of our implementation is an actual place and route result, most of the values in the table are ideal estimates, as few previous studies have provided multi-core evaluation results. These estimations are based on the assumption that circuit resources and throughput are proportional to the number of cores, and that the frequency remains constant as the number of cores increases. The number of cores are determined by the maximum number of pairing processor cores implementable on an xcvu9p in terms of FPGA resources. Formally, the number of cores $\#Cores =$ min($\lfloor 295,560/\#LUTs \rfloor, \lfloor 6,840/\#DSPs \rfloor$), where $295,560$ and $6,840$ are the number of slices and DSPs available on the xcvu9p, respectively. For Devlin's design [11], for example, $\#Cores =$ min($\lfloor 295,560/81,750 \rfloor, \lfloor 6,840/345 \rfloor$) $=$ min$(3, 19) = 3$, where slice resources limits the $\#Cores$.

As shown in the table, our implementations present the best throughput for both BN254 and BLS12_381 pairings. Comparisons with [13], [11], [14], evaluated on FPGAs of the same generation, are important for fair evaluation. [11] uses too many slices (81,750/295,560), which limits the the numbers of cores it can implement to three. Our implementations achieve better latency and several times better throughput compared to [11], [14]. With the SDR close to 43, our architecture can maximize performance in multi-core environments. For the comparison with Opasatian's implementation [13], which is the most comparative to our implementation, we estimated his performance under two conditions (the best and worst cases). Since their study only provides the number of LUTs as resource utilization, we need to estimate their slice utilization

TABLE V
POWER AND ENERGY COMPARISONS OF THE SINGLE-CORE PROPOSED PAIRING PROCESSOR AND PREVIOUS STUDIES.

Design	Curve	Device	ESlices	Freq. [MHz]	Power [W]	Energy / pairing [mJ]
Ours	BLS12_381	xcvu9p	66,552	458	12.03	0.777
[13]	BLS12_381	xcvu13p	84,265*	180	5.98	0.372
[11]	BLS12_381	xcvu37p	96,585	200	7.62*	4.82*
[9]	BLS12_381	Kintex-7	1,006	200	0.079*	2.93*
Ours	BN254	xcvu9p	33,011	590	7.70	0.289
Ours [15]	BN254	xcvu9p	30,622	173	1.33	0.136
[14]	BN254	xczu9eg	5,294	230	0.307	0.293
[10]	BN254	xc6vlx240t	6,860	225	0.389*	0.136*
[41]	BN254	xc6vlx240t	11,355	166	0.476*	0.178*
[12]	BN254	xc7vx690	4,226	180	0.192*	2.74*
[42]	BN254	Core i7-7700	NA	3,600	65†	16.25
[43]	BN254	RTX3060	NA	1,320	170†	3.876

* Estimated value, † Thermal Design Power (TDP)

to calculate their multi-core performance. In the best case, the number of slices utilized is converted as 4 LUTs = 1 slice; $\#Cores = 5$ and the throughput is 80,385. However, typical synthesis results do not follow such an ideal case. For the synthesis results of our implementation, the ratio between LUT and slice is 0.76:1, using this ratio, the $\#Cores$ and throughput of Opasatian's implementation are estimated as 1 and 16,077, respectively.

As a result, our implementations show several times better throughput than the previous studies for a multi-core implementation using the maximum resources of the xcvu9p. Furthermore, we do not consider the negative effects, such as the increase in routing delay caused by multi-core implementation, in the performance estimation for the previous studies. Taking this into account will result in greater performance differences.

3) Energy consumption evaluation: Table V presents the power and energy consumption of the proposed architecture as estimated on Vivado. For previous studies that did not provide power data, we estimated it based on the following CMOS power equation [44, Eq. (5.10)]:

$$P = \alpha f C V^2, \tag{16}$$

where P, α, f, C, and V denote the power, switching rate, clock frequency, switching capacitance of the total circuit, and supply voltage. Assuming that α and V are the same for all implementations enables us to estimate power as $P \propto fC$. We furthermore assume that C is proportional to the circuit area. As a metrics of circuit area to calculate the C values, we use equivalent slices (ESlices $= \#\text{Slices} + 43 \times \#\text{DSPs}$), where 43 (the ratio of the number of slices to the number of DSPs on the xcvu9p) is used as a weight to convert a DSP to slices. As the result, we can estimate the power of previous works as the ratio of frequency and ESlices. Vivado estimated the dynamic power at 458 MHz to be 12.035 W for our BLS12_381 implementation. We can estimate the power consumption of [11] as $12.035 \times \frac{200}{458} \times \frac{96{,}585}{66{,}552} = 7.62$ W. Table V shows that our architecture consumes a significant amount of power, but the energy per pairing is comparative to other studies. Note that Our architecture is designed for cloud FPGAs. Such cloud FPGAs are charged per FPGA; therefore, users do not need to worry about the power consumption, and using as much circuit resources as possible is reasonable.

V. Discussion

A. Performance requirement for the pairing computation

Blockchain-based cryptocurrency is one of the applications that actively uses pairing-based cryptography. Some modern blockchains [25], [24] use the BLS signature, which requires a pairing computation for signature verification, to check the validity of transactions. When a transaction is issued in blockchain-based cryptocurrencies, the payers sign their transaction and sends it to the blockchain's peer-to-peer network. Each peer on the blockchain verifies the signature and adds it to the next block if the verification result is valid. Because signature verification requires pairing, the speed of pairing is directly related to the number of transactions the blockchain can handle. The maximum throughput of 124,216 transactions in our BLS12_381 implementation exceeds the current VISA credit-card transaction capacity of 65,000 transactions per second [45, pp. 8]. If blockchain payments are used as much or more than VISA credit cards, the proposed architecture will be able to handle all transactions. In terms of latency, the proposed architecture will be attractive in the future when blockchain payments are used for latency-critical payments.

B. Evaluation as a PBC accelerator

When we use an FPGA as an accelerator for a general-purpose CPU rather than as a standalone encryption circuit, there is an overhead due to the connection interface between the FPGA and the CPU. We have experimentally estimated this overhead in a system where the FPGA is connected to the server PC via PCIe 3.0 x16. The experimental results show that 2×320 bits and 256×320 bits data transfers take, independent to the transfer direction, about 17 µs and about 22 µs, respectively, where 320 bits is the size of an $\mathbb{F}_p$ element for BN254 pairing. The transfer time t [µs] can be interpolated as $\frac{5}{81.280}x + 16.96$, where x is the number of bits transferred. This implies the CPU execution time of the operating system or application is more dominant than the physical data transfer; the transfer time is not proportional to the amount of data.

Using the above equation, we estimate the performance of the pairing accelerator. Our $\tau = 7$ core BLS12_381 pairing implementation takes 6×7 $\mathbb{F}_p$ elements as an input and 12×7 $\mathbb{F}_p$ elements as an output, where the size of an $\mathbb{F}_p$ element is 436 bits. Assuming the input and output data transfer time is 18 and 19 µs, the latency and throughput will be about 7% worse to 581 µs and 84,337 pairings/s, respectively.

To achieve the same throughput with a CPU, whose single-core throughput is 1,538 as shown in Table III, $\lfloor 84,337/1,538 \rfloor = 52$ CPU cores must be operated in parallel. With some recent high-end CPUs with, it is not impossible to achieve the same throughput as FPGAs; however, it is not reasonable to devote all computational resources to pairing computations because server CPUs must also provide many services other than cryptographic operations. In addition, FPGA accelerators have an advantage in power consumption. Intel Xeon 8592+ CPU with 64 cores have a thermal design power of 350 W, which is tens of times higher than that of FPGAs (see Table V). When all cores are used, the expected performance may not be achieved due to thermal throttling.

C. Applicability to post quantum and other cryptography

Our pairing architecture is designed to efficiently perform the $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ operations, which are the primitive operations for the elliptic curve based pairing computation. Therefore, our architecture can efficiently execute the cryptosystems constructed over $\mathbb{F}_p$, such as isogeny-based cryptography and elliptic curve cryptography. Isogeny-based cryptography is one of the post quantum cryptography (PQC). SQISign [46] is an isogeny-based signature scheme and submitted to NIST as a candidate for PQC standardization. SQISign has the advantage of small key and signature size and the disadvantage of high computational complexity, typically taking tens to thousands of milliseconds on a standard CPU. We believe that FPGA acceleration is a promising solution to this disadvantage, similar to the pairing computation in this paper. Our architecture basically suitable for SQISign that is constructed over Fp2, just modifying instruction scheduling allows us to execute SQISign. Since SQISign uses 252 to 506 bits p depending on the security level, the data path width must be wider to support high security levels (our BL12_381 implementation supports up to 381 bits p).

D. Tamper resistance

The pairing architecture proposed in this paper is primarily designed to maximise throughput, it has no tamper resistance to physical attacks. However, recent studies report that attackers can remotely preform side-channel [47] and fault attacks [48] against FPGAs in the cloud, embedding tamper resistant techniques into our architecture stays remain for one of the future works.

Side-channel attacks include timing attacks, which exploit differences in processing time, and differential power analysis (DPA)-type attacks, which focus on differences in power consumption. Most of the pairing computation in this study are implemented using constant-time algorithms, which are secure against timing attacks because there are no timing differences. The $\mathbb{F}_p$ inversion is the only variable-time part. To make it secure, it is necessary to ensure that the computation always completes in the same cycles by inserting dummy processes.

Against DPA-type attacks and fault attacks, Lashermes et al. propose the use of the randomised projective coordinate [49], where a pairing input is represented as a kind of randomised redundant form. To apply this countermeasure to our architecture, we need to implement a process to add a random number to the input. This can be done by changing the instruction scheduling and is expected to have a time overhead of a few percent at most. The FPGA resource consumption for the random number generator is predicted to be negligible compared to the relatively large multi-core pairing circuit.

VI. Conclusion

First, we proposed an unrolled QPMM algorithm [30] that is suitable for a server-side FPGA, XVU9P, which has DSP48E2 primitives as dedicated multipliers. The proposed method took full advantage of DSP48E2's functions such as the asymmetric multiplier and three-input post-adder, and successfully completed most of the algorithm using only the DSP, achieving the highest throughput (188.98 Gbps), area efficiency (6,873 TP/ESlice), and low SDR (20.16).

We further designed a pairing processor architecture that embeds the proposed modular multiplier. By supporting redundant adders and interleaved executions, the proposed architecture successfully maintained a high frequency and achieved BLS12_381 pairing throughput (15,477 pairings/s). In addition, the proposed pairing architecture has a good SDR (41.24), indicating that it maximizes the performance under multi-core implementation. The multi-core evaluation showed that the throughput of the proposed method was more than 5 times faster than that from previous studies.

References

[1] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and Verifiably Encrypted Signatures from Bilinear Maps," Advances in Cryptology — EUROCRYPT 2003, LNCS, vol. 2656, pp. 416-432, 2003, doi: 10.1007/3-540-39200-9_26.

[2] P. Yanguo, C. Jiangtao, P. Changgen, and Y. Zuobin, "Certificateless public key encryption with keyword search," China Communications, vol. 11, no. 11, pp. 100-113, 2014, doi: 10.1109/CC.2014.7004528.

[3] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, "Faster Explicit Formulas for Computing Pairings over Ordinary Curves," Advances in Cryptology — EUROCRYPT 2011, LNCS, vol. 6632, pp.48-68, 2011, doi: 10.1007/978-3-642-20465-4_5.

[4] B. Razvan, and D. Sylvain, "Updating key size estimations for pairings," Journal of Cryptology, vol. 32, pp. 1298–1336, 2018, doi: 10.1007/s00145-018-9280-5.

[5] X. Wang, Y. Niu, F. Liu, and Z. Xu, "When FPGA Meets Cloud: A First Look at Performance," IEEE Transactions on Cloud Computing, vol. 10, no. 2, pp. 1344-1357, 2022, doi: 10.1109/TCC.2020.2992548.

[6] Valtix, "How and Why on FPGA-based AWS EC2 F1 instances for Cloud Network Security," 2019. [Online] https://valtix.com/blog/valtix_ec2_f1_sc19/

[7] T. Güneysu, and C. Paar, "Ultra High Performance ECC over NIST Primes on Commercial FPGAs," Cryptographic Hardware and Embedded Systems – CHES 2008. LNCS, vol. 5154, pp. 62–78, 2008, doi: 10.1007/978-3-540-85053-3_5

[8] G. X. Yao, J. Fan, R. C. C. Cheung, and I. Verbauwhede, "Faster Pairing Coprocessor Architecture," Pairing-Based Cryptography — Pairing 2012, LNCS, vol. 7708, pp. 160-176, 2012, doi: 10.1007/978-3-642-36334-4_10.

[9] A. Lavice, N. El Mrabet, A. Berzati, J. B. Rigaud, and J. Proy, “Hardware Implementations of Pairings at Updated Security Levels,” Smart Card Research and Advanced Applications — CARDIS 2021, LNCS, vol. 13173, pp. 189-209, 2022, doi: 10.1007/978-3-030-97348-3_11

[10] A. Sghaier, M. Zeghid, L. Ghammam, S. Duquesne, M. Machhout, and H. Y, Ahmed, “High speed and efficient area optimal ate pairing processor implementation over BN and BLS12 curves on FPGA,” Microprocessors and Microsystems, vol. 61, pp. 227-241, 2018, doi: 10.1016/j.micpro.2018.06.001

[11] B. Devlin, “Zcash FPGA acceleration engine,” version 1.4.2 release, Sep. 2019. [Online] https://github.com/ZcashFoundation/zcash-fpga

[12] A. Bag, D. B. Roy, S. Patranabis, and D. Mukhopadhyay, “FlexiPair: An Automated Programmable Framework for Pairing Cryptosystems,” IEEE Transactions on Computers, vol. 71, no. 3, pp. 506-519, 2022, doi: 10.1109/TC.2021.3058345.

[13] A. Opasatian and M. Ikeda, “High-Performance BLS12-381 Pairing Engine on FPGA,” 2023 IEEE 15th International Conference on ASIC (ASICON), pp. 1-4, 2023, doi: 10.1109/ASICON58565.2023.10396122.

[14] M. Bahadori and K. Järvinen, “Compact and Programmable yet High-Performance SoC Architecture for Cryptographic Pairings,” 2020 30th International Conference on Field-Programmable Logic and Applications (FPL), pp. 176-184, 2020, doi: 10.1109/FPL50879.2020.00038.

[15] J. Sakamoto, Y. Nagahama, D. Fujimoto, Y. Okuaki, and T. Matsumoto, “Low-Latency Pairing Processor Architecture Using Fully-Unrolled Quotient Pipelining Montgomery Multiplier,” 2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST), pp. 1-6, 2019, doi: 10.1109/AsianHOST47458.2019.9006671.

[16] D. Suzuki, “How to Maximize the Potential of FPGA Resources for Modular Exponentiation,” Cryptographic Hardware and Embedded Systems — CHES 2007, LNCS, vol. 4727, pp. 272-288, 2007. doi: 10.1007/978-3-540-74735-2_19

[17] G. Gallin, and A. Tisserand, “Generation of Finely-Pipelined GF(P) Multipliers for Flexible Curve Based Cryptography on FPGAs,” IEEE Transactions on Computers, vol. 68, no. 11, pp. 1612-1622, 2019, doi: 10.1109/TC.2019.2920352

[18] F. Vercauteren, “Optimal Pairings,” IEEE Transactions on Information Theory, vol. 56, no. 1, pp. 455-461, 2010, doi: 10.1109/TIT.2009.2034881.

[19] P. S. L. M. Barreto, and M. Naehrig, “Pairing-Friendly Elliptic Curves of Prime Order,” Selected Areas in Cryptography, SAC 2005, LNCS, vol. 3897, pp.319-331, 2006, doi: 10.1007/11693383_22

[20] P. S. L. M. Barreto, B. Lynn, and M. Scott, “Constructing Elliptic Curves with Prescribed Embedding Degrees,” Security in Communication Networks SCN 2002, LNCS, vol. 2576, pp. 257-267, 2003. doi: 10.1007/3-540-36413-7_19

[21] Y. Sakemi, T. Kobayashi, T. Saito, and R. S. Wahby, “draft-irtf-cfrg-pairing-friendly-curves-11,” 2022. [Online] https://www.ietf.org/archive/id/draft-irtf-cfrg-pairing-friendly-curves-11.txt.

[22] D. F. Aranha, P. S. L. M. Barreto, P. Longa, and J. E. Ricardini , “The Realm of the Pairings,” Selected Areas in Cryptography – SAC 2013, LNCS, vol. 8282, pp. 3-25, 2014, doi: 10.1007/978-3-662-43414-7_1

[23] D. Boneh, B. Lynn, and H. Shacham, “Short Signatures from the Weil Pairing,” Advances in Cryptology — ASIACRYPT 2001, LNCS, vol. 2248, pp. 513-532, 2001. doi: 10.1007/3-540-45682-1_30

[24] T. Hanke, M. Movahedi, and D. Williams, “DFINITY Technology Overview Series Consensus System Rev. 1,” 2018. [Online] https://arxiv.org/abs/1805.04548.

[25] R. Jordan, “Ethereum 2.0 Development Update #17 — Prysmatic Labs,” 2018. [Online] https://medium.com/prysmatic-labs/ethereum-2-0-development-update-17-prysmatic-labs-ed5bcf82ec00.

[26] C. Gentry and Z. Ramzan, “Identity-Based Aggregate Signatures,” Public Key Cryptography — PKC 2006, LNCS, vol. 3958, pp. 27-273, 2006. doi: 10.1007/11745853_17

[27] L. Shen, J. Ma, X. Liu, F. Wei, and M. Miao, “A Secure and Efficient ID-Based Aggregate Signature Scheme for Wireless Sensor Networks,” IEEE Internet of Things Journal, vol. 4, no. 2, pp. 546-554, 2017, doi: 10.1109/JIOT.2016.2557487.

[28] P. Barrett, “Implementing the Rivest Shamir and Adleman Public Key Encryption Algorithm on a Standard Digital Signal Processor,” Advances in Cryptology — CRYPTO’ 86, LNCS, vol. 263, pp. 311-323, 2000, doi: 10.1007/3-540-47721-7_24

[29] P. Montgomery, “Modular Multiplication Without Trial Division,” Math. Computation, vol. 44, pp. 519–521, 1985.

[30] H. Orup, “Simplifying quotient determination in high-radix modular multiplication,” the 12th Symposium on Computer Arithmetic, pp. 193-199, 1995. doi: 10.1109/ARITH.1995.465359.

[31] E. Savaş, and C. K. Koç, “Montgomery inversion,” Journal of Cryptographic Engineering, vol. 8, pp. 201–210, 2017. doi: 10.1007/s13389-017-0161-x

[32] Xilinx, “Virtex UltraScale+ FPGA Data Sheet: DC and AC Switching Characteristics,” DS923 (v1.19), 2021. [Online] https://docs.xilinx.com/v/u/en-US/ds923-virtex-ultrascale-plus

[33] Y. Ma, Z. Liu, W. Pan, and J. Jing , “A High-Speed Elliptic Curve Cryptographic Processor for Generic Curves over GF(p),” Selected Areas in Cryptography — SAC 2013, LNCS, vol. 8282, pp. 421-437, 2014. doi: 10.1007/978-3-662-43414-7_21

[34] L. Noyez, N. E. Mrabet, O. Potin, and P. Veron, “Montgomery Multiplication Scalable Systolic Designs Optimized for DSP48E2,” ACM Trans. Reconfigurable Technol. Syst. 17, 1, Article 9, pp. 1-31, 2024, doi: 10.1145/3624571.

[35] E. Öztürk, “Design and Implementation of a Low-Latency Modular Multiplication Algorithm,” IEEE Transactions on Circuits and Systems I: Regular Papers, vol. 67, no. 6, pp. 1902-1911, 2020, doi: 10.1109/TCSI.2020.2966755.

[36] A. A. H. Abd-Elkader, M. Rashdan, E. -S. A. M. Hasaneen, and H. F. A. Hamed, ”FPGA-Based Optimized Design of Montgomery Modular Multiplier,” IEEE Transactions on Circuits and Systems II: Express Briefs, vol. 68, no. 6, pp. 2137-2141, 2021, doi: 10.1109/TCSII.2020.3040665.

[37] D. B. Roy, D. Mukhopadhyay M. Izumi, and J. Takahash, “ Tile Before Multiplication: An Efficient Strategy to Optimize DSP Multiplier for Accelerating Prime Field ECC for NIST Curves,” 51st Annual Design Automation Conference DAC ’14, pp. 1-6, 2014.

[38] D. B. Roy, and D. Mukhopadhyay, “High-Speed Implementation of ECC Scalar Multiplication in GF(p) for Generic Montgomery Curves,” IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 27, no. 7, pp. 1587-1600, 2019, doi: 10.1109/TVLSI.2019.2905899.

[39] D. Mukhopadhyay, and D. B. Roy, “Revisiting FPGA Implementation of Montgomery Multiplier in Redundant Number System for Efficient ECC Application in GF(p),” 28th International Conference on Field Programmable Logic and Applications (FPL), pp. 323-3233, doi: 10.1109/FPL.2018.00061.

[40] Z. Ni, D. -E. -S. Kundi, M. O’Neill, and W. Liu, “High-Performance Systolic Array Montgomery Multiplier for SIKE,” 2021 IEEE International Symposium on Circuits and Systems (ISCAS), pp. 1-5, 2021, doi: 10.1109/ISCAS51556.2021.9401062.

[41] S. Ghosh, I. Verbauwhede, and D. Roychowdhury, “Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform,” Pairing-Based Cryptography — Pairing 2012, LNCS, vol. 7708, pp. 141-159, 2013, doi: 10.1007/978-3-642-36334-4_9

[42] Herumi, “mcl: A portable and fast pairing-based cryptography library,” [Online] https://github.com/herumi/mcl.

[43] X. Hu, D. He, M. Luo, C. Peng, Q. Feng, and X. Huang, “High-Performance Implementation of the Identity-Based Signature Scheme in IEEE P1363 on GPU,” ACM Trans. Embed. Comput. Syst. 22, 2, Article 25, pp. 1–35, 2023, doi: 10.1145/3564784.

[44] N. H.E. Weste and D. Harris, “CMOS VLSI design : a circuits and systems perspective 4th ed.,” Addison-Wesley, 2010.

[45] Visa, “VISA Annual Report 2016,” [Online] https://s1.q4cdn.com/050606653/files/doc_financials/annual/Visa-2016-Annual-Report.pdf

[46] L. D. Feo, D. Kohel, A. Leroux, C. Petit, and B. Wesolowski, “SQISign: Compact Post-quantum Signatures from Quaternions and Isogenies,” Advances in Cryptology – ASIACRYPT 2020, LNCS, vol. 12491, pp. 64-93, 2020, doi: 10.1007/978-3-030-64837-4_3

[47] M. Zhao and G. E. Suh, “FPGA-Based Remote Power Side-Channel Attacks,” 2018 IEEE Symposium on Security and Privacy (SP), pp. 229-244, 2018, doi: 10.1109/SP.2018.00049.

[48] J. Krautter, D. R. E. Gnad, and M. B. Tahoori, “FPGAhammer: Remote Voltage Fault Attacks on Shared FPGAs, suitable for DFA on AES”, IACR Transactions on Cryptographic Hardware and Embedded Systems, vol. 2018, no. 3, pp. 44–68, 2018, doi: 10.13154/tches.v2018.i3.44-68.

[49] T. H. Kim, T. Takagi, DG. Han, H. W. Kim, and J. Lim, “Side Channel Attacks and Countermeasures on Pairing Based Cryptosystems over Binary Fields,” in Cryptology and Network Security. CANS 2006, LNCS, vol. 4301, pp. 168-181, 2006, doi: 10.1007/11935070_11.