Compact and Programmable yet High-Performance SoC Architecture for Cryptographic Pairings

Milad Bahadori and Kimmo Järvinen
University of Helsinki, Department of Computer Science, Helsinki, Finland
{milad.bahadori, kimmo.u.jarvinen}@helsinki.fi

***Abstract*—Cryptographic pairings are important primitives for many advanced cryptosystems. Efficient computation of pairings requires the use of several layers of algorithms as well as optimizations in different algorithm and implementation levels. This makes implementing cryptographic pairings a difficult task particularly in hardware. Many existing hardware implementations fix the parameters of the pairing to improve efficiency but this significantly limits the generality and practicality of the solution. In this paper, we present a compact and programmable yet high-performance architecture for programmable system-on-chip platforms designed for efficient computation of different cryptographic pairings. We demonstrate with real hardware that this architecture can compute optimal ate pairings on a Barreto-Naehrig curve with 126-bit security in 2.18 ms in a Xilinx Zynq-7020 device and occupies only about 3200 slices, 36 DSPs, and 18 BRAMs. We also show that the architecture can support different types of pairings via microcode updates and can be implemented on other reprogrammable devices with very minor modifications.**

***Index Terms*—Cryptographic pairing, system-on-chip, HW/SW codesign, FPGA, optimal Ate pairing, Barreto-Naehrig curves.**

I. Introduction

In cryptology, bilinear pairings were first used for cryptanalysis [1] but have been later used in building various advanced cryptosystems, such as tripartite key exchange [2], identity-based encryption [3], short signatures [4], attribute-based encryption [5], [6], searchable encryption [7], functional encryption [8], [9], etc. This has created a need for efficiently computable cryptographic pairings and resulted in significant amounts of research in improving efficiency of pairings on both algorithm and implementation levels. On the theoretical side, notable research results include various types of pairing algorithms (e.g., Tate [10]–[12], η_T [13], ate [14], R-ate [15], and optimal ate [16] pairings) and pairing-friendly elliptic curves (in particular, Barreto-Naehrig (BN) curves [17]).

On the implementation side, efficient implementations have been presented both in software [18]–[22] and hardware [23]–[37], the latter including both Field Programmable Gate Arrays (FPGAs) and Application Specific Integrated Circuits (ASICs). Pairings are very complicated operations including multiple layers of algorithms (e.g., [22] utilizes 31 algorithms to compute an optimal ate pairing) and efficient pairing computations require careful choices of parameters and algorithmic tricks. Consequently, their implementation is notoriously difficult and laborious, especially, in hardware.

While software provides natural flexibility and allows support for multiple pairings as well as easily updating pairing algorithms, hardware is significantly more rigid. Although fixing parameters leads to a more efficient implementation, it may come with a significant penalty in practical feasibility because it reduces flexibility regarding types of pairings and curves and hinders the adaptation of new algorithms. In theory, flexibility could be provided with reprogrammable hardware but, in practice, it may be hard because pairings are complicated algorithms and designing separate implementations for all pairing types and parameter sets would be a daunting task. Hence, there is a clear need for flexible high-performance implementations that can be used for efficiently computing different cryptographic pairings. Typically, pairings are only a part of a cryptosystem and also other operations must be supported by an implementation in order to realize the cryptosystem (e.g., identity-based encryption, searchable encryption, or functional encryption schemes). Hence, an implementation of pairings should be compact and achieve a good speed-area tradeoff.

Hardware/Software (HW/SW) codesign paradigm is suitable for pairing computations and their use in larger cryptosystems because complicated control flows can be implemented in software while still receiving the benefits of hardware acceleration effectively with efficient yet compact accelerator cores. This is particularly due to the fact that complicated state machines required for controlling complex pairing computations are easy and efficient to implement in software whereas they incur significant area overheads in hardware. A HW/SW codesign is also scalable in the sense that it can be extended with additional cores for parallel pairings and/or other operations needed by the cryptosystem. In this paper, we will focus on programmable System-on-Chip (SoC) platforms (e.g., Xilinx Zynq SoCs) that realize the HW/SW codesign paradigm with hardwired processors (typically ARM cores) and reprogrammable hardware (i.e., FPGAs).

To keep the discussion concise and clear, we focus particularly on optimal ate pairings [16] over BN curves [17] and the specific parameters used by Beuchat et al in [22]. Nevertheless, we emphasize that the implementation is generic and can be used for implementing various pairings on different curves.

In this paper, we provide the following contributions:

- We describe a compact programmable SoC architecture for cryptographic pairings that achieves high performance and very good speed-area tradeoff. The architecture is optimized for the resources of modern reprogrammable SoCs such as DSPs, BlockRAMs, and hard ARM cores.

Algorithm 1: Optimal ate pairing over BN curves.

```
   Input: $P \in \mathbb{G}_1$ and $Q \in \mathbb{G}_2$.
   Output: $a_{\text{opt}}(Q, P) = f$, where $f \in \mathbb{F}_{p^{12}}$.
   Constant: $s = 6t + 2 = \sum_{i=0}^{L-1} s_i\, 2^i$, where
             $s_i \in \{-1, 0, +1\}$.
1  $T \leftarrow Q$, $f \leftarrow 1$
2  for $i = L - 2$ to $0$ do
3      $f \leftarrow f^2 \cdot l_{T,T}(P)$; $T \leftarrow 2T$
4      if $s_i \neq 0$ then
5          $f \leftarrow f \cdot l_{T,s_iQ}(P)$; $T \leftarrow T + s_i\, Q$
6  $Q_1 \leftarrow \pi_p(Q)$; $Q_2 \leftarrow -\pi_{p^2}(Q)$
7  $f \leftarrow f \cdot l_{T,Q_1}(P)$; $T \leftarrow T + Q_1$
8  $f \leftarrow f \cdot l_{T,Q_2}(P)$; $T \leftarrow T + Q_2$
9  $f \leftarrow f^{(p^{12}-1)/r}$
10 return $f$
```

- The architecture supports microcode updates that can be used for supporting different cryptographic pairing algorithms, curves, and other parameters with the same accelerator architecture. This makes our architecture significantly more viable for practical deployment than hardware implementations with fixed parameters.
- We evaluate the proposed HW/SW codesign system on real hardware using Avnet Zedboard including a Xilinx Zynq-7020 programmable SoC chip and showcase the above-mentioned benefits.

The rest of this paper is organized as follows. We briefly survey the relevant algorithmic background in Section II. We present the architecture of our implementation and the computation procedures in Section III followed by results and analysis in Section IV. Finally, we end the paper by drawing conclusions in Section V.

II. Preliminaries of Pairing

A cryptographic pairing is a bilinear map $\mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_3$ where $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups and $\mathbb{G}_3$ is a multiplicative group. In the context of optimal ate pairings on BN curves, $\mathbb{G}_1$ and $\mathbb{G}_2$ are additive groups of points on elliptic curves $E(\mathbb{F}_p)$ and $E(\mathbb{F}_{p^k})$ and $\mathbb{G}_3$ is the multiplicative group of $\mathbb{F}_{p^k}$. The parameters must be chosen so that discrete logarithms in all three groups are infeasible; e.g., for approximately 128-bit security level, we need a 256-bit prime p and $k = 12$.

The algorithm for computing an optimal ate pairing over BN curves is given in Alg. 1. The two main operations in the algorithm are the Miller loop in lines 2–5 and the final exponentiation in line 9. The former consists of elliptic curve arithmetic in $E(\mathbb{F}_{p^2})$ and line evaluations in $\mathbb{F}_{p^{12}}$ that can be interleaved. The latter is an exponentiation in $\mathbb{F}_{p^{12}}$ that can be decomposed into $f^{(p^6-1)(p^2+1)(p^4-p^2+1)/r}$, of which the two first terms can be efficiently computed with Frobenius operators and conjugations. The last term is called the hard part and is computationally the most demanding part.

We demonstrate our design for computing Alg. 1 by using the subalgorithms from [22]. They used $t = 2^{62} - 2^{54} + 2^{44}$ that enables efficient computation of the Miller loop and the hard part of the final exponentiation while providing 126-bit security level. The primes p and r are as follows: $p = 36t^4 + 36t^3 + 24t^2 + 6t + 1$ and $r = 36t^4 + 36t^3 + 18t^2 + 6t + 1$. In [22], $\mathbb{F}_{p^{12}}$ is represented as a tower extension field with the following irreducible binomials:

$$\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta), \text{where } \beta = -5 \tag{1}$$

$$\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi), \text{where } \xi = u \tag{2}$$

$$\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - v). \tag{3}$$

Consequently, arithmetic operations in the above fields are computed with series of operations in $\mathbb{F}_p$. In particular, the Karatsuba-like construction allows multiplications in the quadratic extension fields $\mathbb{F}_{p^2}$ and $\mathbb{F}_{p^{12}}$ to be computed with three multiplications (and additions/subtractions) in the underlying fields $\mathbb{F}_p$ and $\mathbb{F}_{p^6}$, respectively. Multiplications in $\mathbb{F}_{p^6}$ require six multiplications in $\mathbb{F}_{p^2}$ [22]. Lines 3, 5, 7, and 8 are computed using formulae from [18], [38]. Line 6 requires only three multiplications and two negations in $\mathbb{F}_p$. Line 9 follows the ideas of [39] and consists of multiple low-level algorithms and optimizations.

III. Architecture and Implementation

In this section, we present our architecture for pairings using the HW/SW codesign approach in a programmable SoC. Most of the existing hardware-based pairing implementations have focused on maximizing the speed at the expense of resource utilization and programmability. The few flexible designs that support different pairings and parameters are significantly slower. The HW/SW codesign approach allows an efficient tradeoff combining high performance with low resource usage and flexibility. This is particularly true for pairing computations where the main difficulty in this respect is the high number of different algorithms that must be supported but where computations mostly rely on the same low-level operations (i.e., $\mathbb{F}_p$ arithmetic).

A. High-Level HW/SW Codesign

Our architecture is constructed as a generic HW/SW codesign and can be instantiated in various programmable SoCs with minor modifications. However, in this paper, we consider mainly instantiations in Xilinx all-programmable SoCs because we use Avnet ZedBoard and Xilinx ZCU102 evaluation kits for prototyping. We will refer to the specific features of those programmable SoCs whenever such a distinction is required. Also, to provide programmability and to decrease resource utilization, the HW part of our architecture uses a microprogramming appraoch instead of implementing hardwired Finite State Machines (FSMs) for the specific algorithms of pairing computations. Because microprogramming provides flexibility, scalability, and programmability combined with a small area footprint that would be hard to achieve with specific FSMs in hardware.

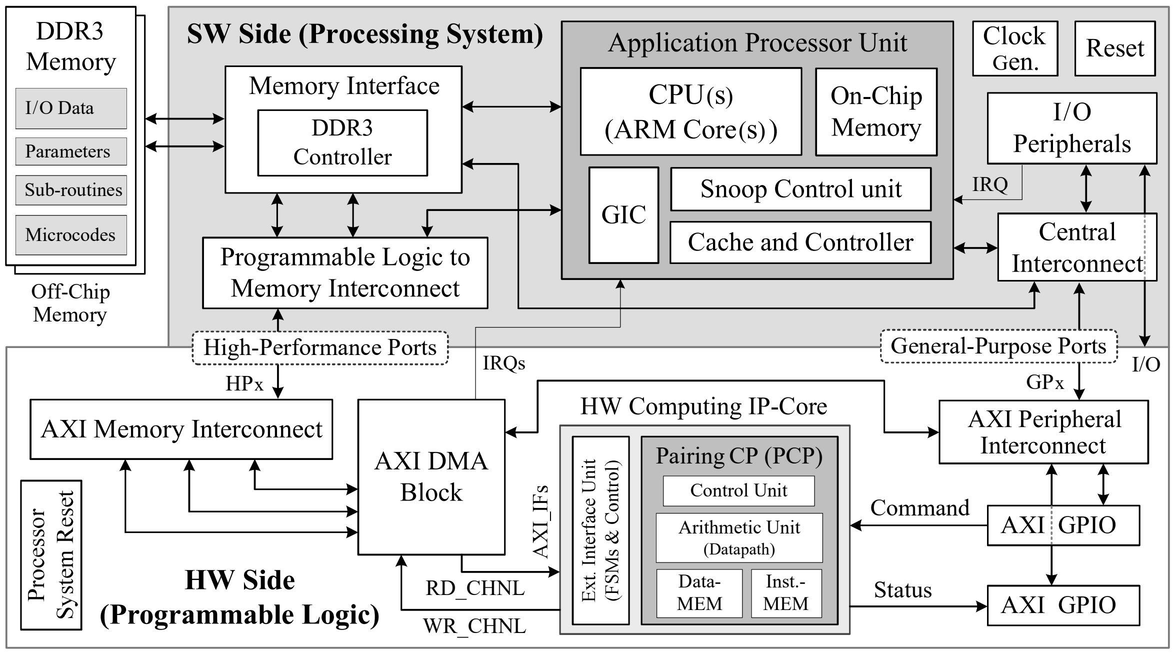

Fig. 1. High level architecture of the HW/SW codesign for the pairing.

Fig. 1 illustrates the high-level architecture of the HW/SW codesign which is divided into two main parts including SW and HW sides (called Processing System (PS) and Programmable Logic (PL) in Xilinx terminology, respectively). The SW side consists of ARM core(s), on-chip and off-chip (i.e., DDR3) memories, and other interconnection and control. The HW side consists of Pairing Cryptography Processor (PCP) and supporting modules (i.e., Xilinx IP cores including Direct Memory Access (DMA), memory and peripheral interconnects, General Purpose Input/Output (GPIO), and processor system reset). The data and control communications between the SW and HW sides are based on the capabilities of the specific programmable SoC, and we use the Advanced Extensible Interface (AXI) High Performance (HP) and General Purpose (GP) interfaces of Xilinx SoCs. The HP interface is employed for high-performance transfer of data and microcodes, and the GP interface is used for transferring commands and status (see Fig. 1). The SW side is responsible for controlling the HW side and external peripherals. Specifically, the SW side performs the high-level control and managing of the execution-flow of the pairing computation. These operations include sending and receiving data and microcode packets to/from the PCP, issuing commands to the PCP, offline and online programming of the PCP (by the microcodes) and other modules in the HW side, receiving the status of the PCP and other modules from the HW side, and making control decisions based on the received status. As shown in Fig. 1, all modules in the HW side are connected in an AXI-based structure. The high-performance data and microcodes communication between the SW side and the PCP of the HW side is done via the HP_x interface that connects to the AXI memory interconnect block which further connects to the PCP core via an AXI DMA block. Furthermore, the command and status communication is handled via the AXI peripheral interconnect block in the HW side. It is also used for controlling the AXI DMA block used for high-speed data and microcodes communications.

B. Pairing Cryptography Processor (PCP)

The cost of a pairing computation is generally expressed by the total number of required field operations (i.e., multiplications, additions/subtractions, constant-multiplications, and inversions). Moreover, the efficiencies of the architecture and the scheduling technique of field operations are the main factors that determine the overall performance of a pairing implementation [37]. The main objective in designing the PCP is to achieve a good trade-off between programmability, speed, and area requirements and to efficiently utilize the resources of modern FPGAs (e.g., DSPs and BRAMs) in implementing base field arithmetic (i.e., arithmetic in $\mathbb{F}_p$). Because the tower extension field arithmetic is ultimately based on $\mathbb{F}_p$ arithmetic, this allows us to efficiently implement different arithmetic operations in $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$, and $\mathbb{F}_{p^{12}}$ (tower field arithmetic).

Fig. 2 depicts the architecture of the PCP, which contains external interface, arithmetic (datapath), control, Data Memory (DMEM), and Instruction Memory (IMEM) units. The external interface unit is used for command, status, data, and microcode communication with the external modules. The IMEM contains a 1024×72-bit simple dual-port RAM and a controller for different address branch scenarios. IMEM stores microcodes for algorithm(s) that are run in the PCP. Each instruction in the microcodes consists of several fields that apply the required commands to the corresponding units for a working cycle of the PCP. The IMEM is partitioned into

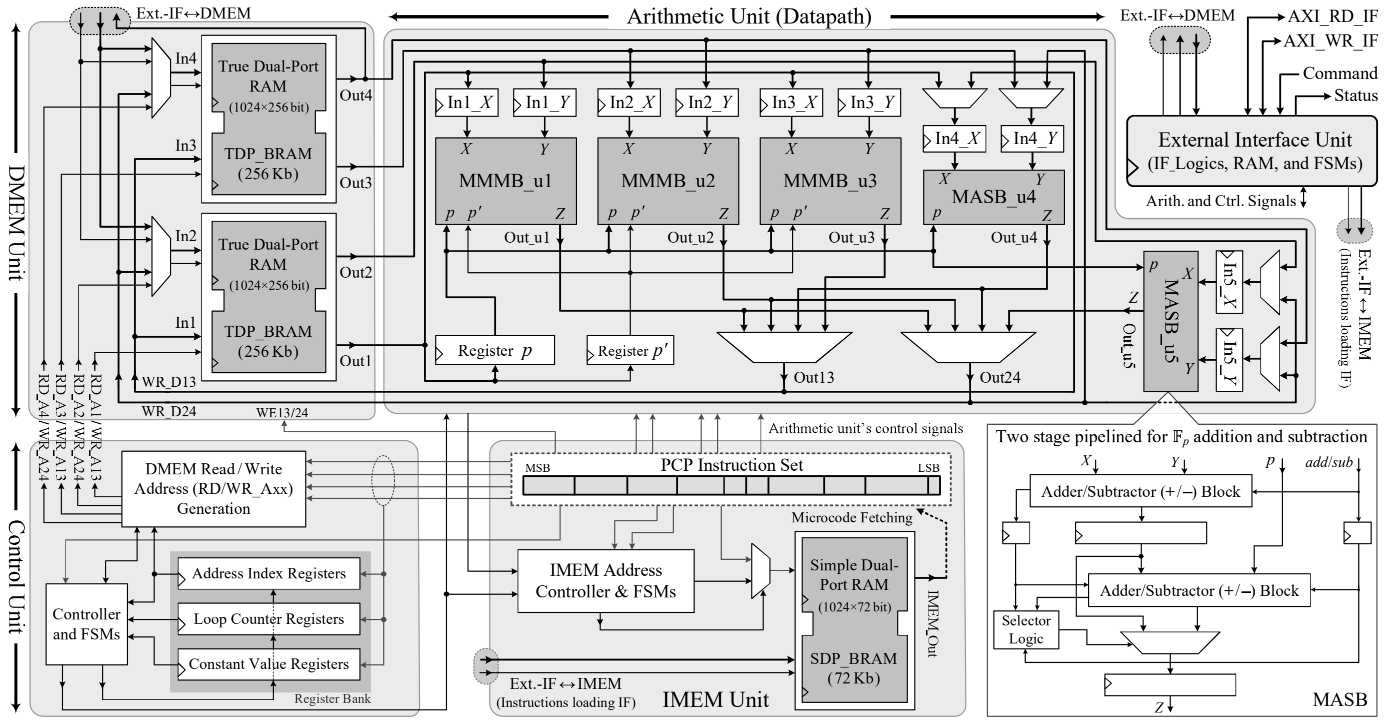
Fig. 2. Architecture details of the PCP core including external interface, DMEM, datapath, IMEM, and control units. Structure of the MASB is described.

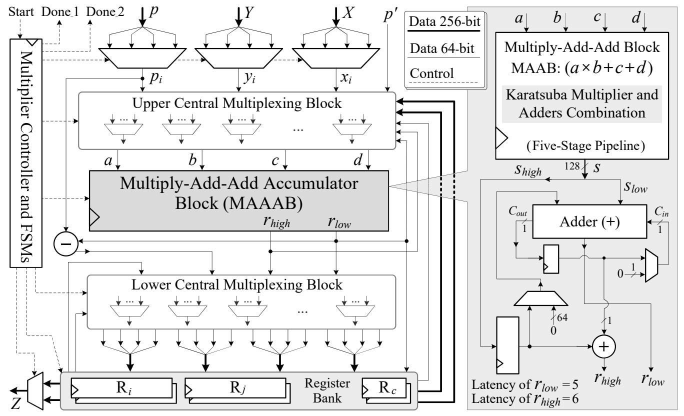
Fig. 3. Architecture details of the MMMB.

32 segments (i.e., $32 \times 2.25\text{Kb} = 72\text{Kb}$), where each segment can be loaded separately via the external interface unit during the runtime. In addition, full microcode loading of the IMEM can be done by the SW side directly during the runtime.

The control unit generates addresses for DMEM and makes decisions for loop iterations and conditional statements. The inputs and outputs of the arithmetic unit are connected to DMEM, which stores data that is required during an algorithm run. DMEM is a duplicated 1024×256-bit true dual-port RAM with two independent read and write ports and supports “4-read”, “2-write”, or “2-read and 1-write” operations from/to DMEM. This facilitates efficient scheduling and parallelization of $\mathbb{F}_p$ arithmetic. DMEM is also interfaced with the external interface unit for communicating data with the SW side.

1) Arithmetic Unit: The datapath is shown in the top right corner of Fig. 2 and it supports arithmetic in $\mathbb{F}_p$ with arbitrary up to 256-bit primes p (i.e., up to 128-bit security). It consists of three parts: source registers, arithmetic blocks, and output selectors. The arithmetic blocks comprise three Montgomery Modular Multiplier Blocks (MMMBs) and two Modular Adder/Subtractor Blocks (MASBs) and they can operate in parallel and independently of each other. The inputs of all arithmetic blocks can be loaded from DMEM but the inputs of MASBs can be additionally loaded from the outputs of the arithmetic blocks. This arrangement together with the multi-read/write feature of DMEM allows efficient computation of tower extension field arithmetic. E.g., $\mathbb{F}_{p^2}$ multiplication requires three $\mathbb{F}_p$ multiplications, which can be computed as follows: $[(a_0 \times b_0), (a_1 \times b_1), ((a_0 + a_1) \times (b_0 + b_1))] \equiv [(\text{Out}_1 \times \text{Out}_2), (\text{Out}_3 \times \text{Out}_4), ((\text{Out}_1 + \text{Out}_3) \times (\text{Out}_2 + \text{Out}_4))]$. The modulus p and the precomputed Montgomery constant p' are registered into the arithmetic unit.

a) MASB: The structure of MASB with a two-stage pipeline is also illustrated in Fig. 2. Addition and subtraction in $\mathbb{F}_p$ can be realized by two consecutive adder/subtractor circuits which produce the result in two cycles. Due to the pipeline, its throughput is one $\mathbb{F}_p$ addition/subtraction per cycle. Applying two MASBs and connecting the outputs of the datapath back to its inputs facilitates efficient field arithmetic operations such as $\mathbb{F}_{p^2}$ addition/subtraction/negation, $\mathbb{F}_{p^2}$ multiplication/squaring, and multiplications by small constants.

b) MMMB: Fig. 3 shows the structure details of MMMB for computing $\mathbb{F}_p$ multiplications/squarings. It contains three nested parts which are organized bottom-up as a Multiply-

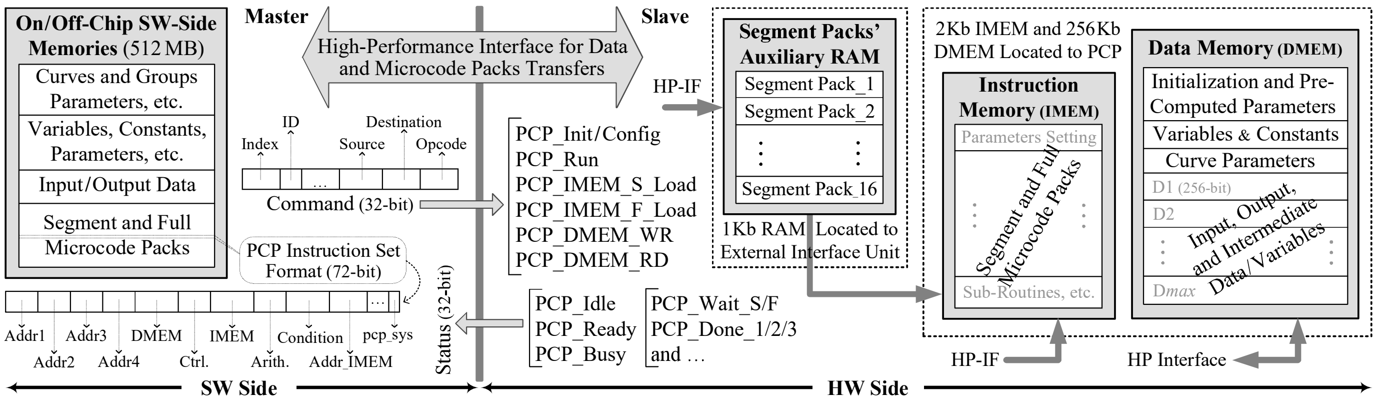
On/Off-Chip SW-Side Memories (512 MB)
Curves and Groups Parameters, etc.
Variables, Constants, Parameters, etc.
Input/Output Data
Segment and Full Microcode Packs
Master
High-Performance Interface for Data and Microcode Packs Transfers
Slave
Index
ID
Source
Destination
Opcode
Command (32-bit)
PCP Instruction Set Format (72-bit)
PCP_Init/Config
PCP_Run
PCP_IMEM_S_Load
PCP_IMEM_F_Load
PCP_DMEM_WR
PCP_DMEM_RD
Addr1
Addr2
Addr3
Addr4
DMEM
Ctrl.
IMEM
Arith.
Condition
Addr_IMEM
pcp_sys
Status (32-bit)
PCP_Idle
PCP_Ready
PCP_Busy
PCP_Wait_S/F
PCP_Done_1/2/3
and …
HP-IF
Segment Packs' Auxiliary RAM
Segment Pack_1
Segment Pack_2
Segment Pack_16
1Kb RAM Located to External Interface Unit
2Kb IMEM and 256Kb DMEM Located to PCP
Instruction Memory (IMEM)
Parameters Setting
Segment and Full Microcode Packs
Sub-Routines, etc.
Data Memory (DMEM)
Initialization and Pre-Computed Parameters
Variables & Constants
Curve Parameters
D1 (256-bit)
D2
Input, Output, and Intermediate Data/Variables
Dmax
HP-IF
HP Interface
SW Side
HW Side

Fig. 4. Memory taxonomy details of the SW and HW sides, SW/HW interaction principles, and PCP instruction set format of the HW/SW codesign system.

Add-Add Block (MAAB), a Multiply-Add-Add-Accumulator Block (MAAAB), and the overall structure of MMMB. MAAB is the primary computation block in the datapath and consists of a 64×64 bit Karatsuba multiplier (constructed from three parallel 32×32 bit multipliers) combined with adders to compute $a \times b + c + d$ (all 64-bit values) in a five-stage pipeline. MAAB consumes most of the FPGA resources, has the highest dynamic power consumption, and also contains the critical path of PCP. In order to maximize its efficiency, it is implemented using the DSP slices. In the next part, MAAB is complemented with an accumulation operation (i.e., MAAAB). The lower part of the MAAB result is accumulated with the previous higher part as well as with the previous most significant bit of the accumulation result (i.e., the input carry). The output carry and the higher part of the MAAB result are stored for the next accumulation (see Fig. 3). The latencies for computing r_{low} and r_{high} are five and six clock cycles, respectively. This accumulation method and the one clock cycle difference between r_{low} and r_{high} are essential for efficient implementation of high-radix Montgomery modular multiplication algorithm [40]. Finally, in the top part, MAAAB (as the main computing core) as well as multiplexers, registers, and FSMs are used for implementing radix-2^{64} Montgomery modular multiplication [40]. MMMB computes a multiplication/squaring in $\mathbb{F}_p$ with a total latency of 43 clock cycles, but a new multiplication/squaring can be started already after 38 clock cycles due to the pipelined scheme.

C. Pairing Computations with the HW/SW Codesign

1) Working Principle and Scheduling of the Architecture: The initialization step configures both the SW and HW sides of the HW/SW codesign and must be done only once for every pairing algorithm and curve parameter set. It includes loading all inputs and curve parameters into DMEM of the PCP core in the HW side. To implement a specific pairing algorithm, an in-depth analysis of the algorithm is performed and all algorithms of the pairing are translated into microcodes (i.e., several segments and/or full sub-routine packs). The microcodes are sequences of instructions for different units of the PCP core. Each instruction set consists of fields such as arithmetic, control, next IMEM address, DMEM address values, DMEM, and IMEM fields. These fields apply all required controlling signals for the units for a working cycle of the PCP core. The microcodes are generated by hand through a customized platform and scripts. In this architecture, each instruction set has 72-bit length and it is divided to 14 fields. The microcodes are stored in the (off/on-chip) SW side memory (i.e., DRR3). Whenever a (set of) particular computation(s) needs to be executed in PCP, then the corresponding microcode(s) are loaded into IMEM by SW side through the external interface unit, as explained before. According to the aforementioned explanations, details of the memory taxonomy in the SW and HW sides, SW/HW interaction principles, and the PCP instruction set format are described in Fig. 4. Obviously, the efficiency of microcodes for computing tower extension field arithmetic greatly determines the overall performance of a pairing computation and, therefore, special care should be taken in scheduling operations and maximizing the utilization of the datapath for these operations.

Fig. 5 illustrates how to efficiently implement and schedule the tower extension field arithmetic (from $\mathbb{F}_p$ to $\mathbb{F}_{p^{12}}$) on the BN_{126} curve [22], which is the main focus of this paper. On the top, it shows how to maximize the usage and scheduling of the datapath for $\mathbb{F}_p$ arithmetic (i.e., multiplications/squarings, additions/subtractions) by utilizing parallelism and pipelining. The datapath effectively hides the costs of additions/substractions as they can be computed simultaneously with multiplications and this can be utilized for efficient computation of tower field arithmetic. In the middle, Fig. 5 depicts the realization of $\mathbb{F}_{p^2}$ arithmetic using $\mathbb{F}_p$ arithmetic and shows that a new $\mathbb{F}_{p^2}$ multiplication/squaring can be computed after 38 clock cycles and, also, that up to eleven $\mathbb{F}_{p^2}$ additions/subtractions can be done during each $\mathbb{F}_{p^2}$ multiplication/squaring. The $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ operations are further used for implementing $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$, and $\mathbb{F}_{p^{12}}$ arithmetic (addition, subtraction, negation, constant-multiplication, multiplication, squaring, exponentiation, and inversion).

2) Optimal Ate Pairing Computation Steps: Implementation of optimal ate pairing algorithm (Alg. 1) in our HW/SW codesign consists of three levels. The first level are the im-

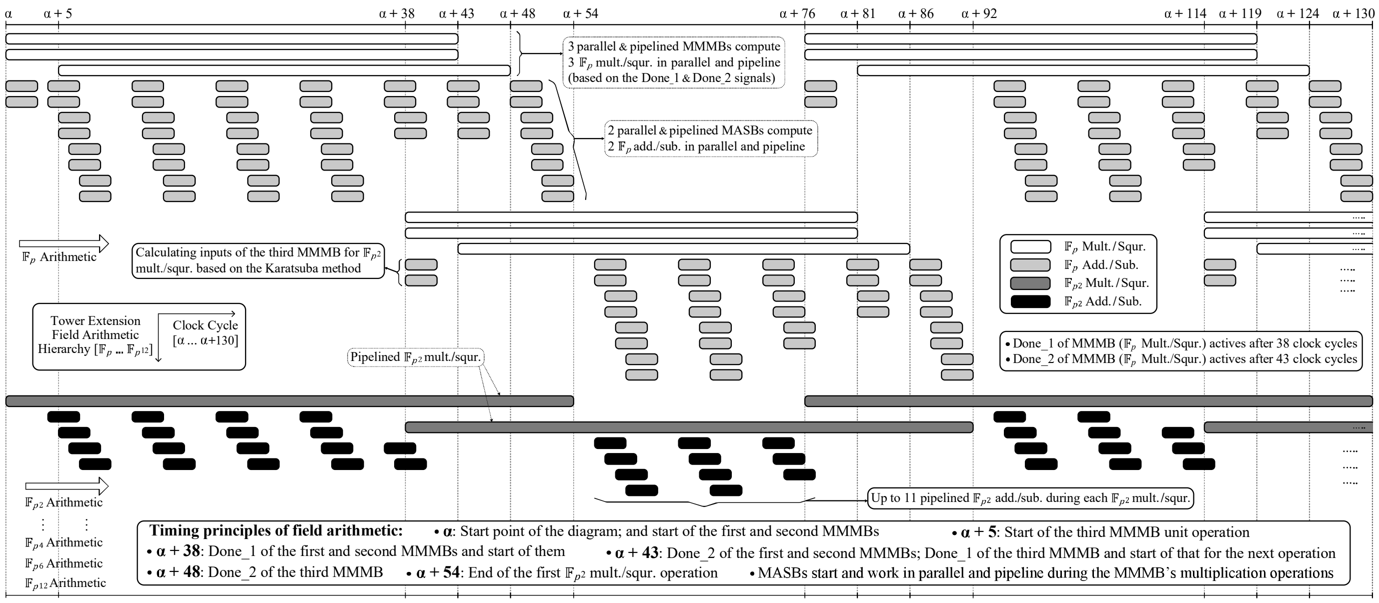
α
α + 5
α + 38
α + 43
α + 48
α + 54
α + 76
α + 81
α + 86
α + 92
α + 114
α + 119
α + 124
α + 130
3 parallel & pipelined MMMBs compute 3 $\mathbb{F}_p$ mult./squr. in parallel and pipeline (based on the Done_1 & Done_2 signals)
2 parallel & pipelined MASBs compute 2 $\mathbb{F}_p$ add./sub. in parallel and pipeline
$\mathbb{F}_p$ Arithmetic
Calculating inputs of the third MMMB for $\mathbb{F}_{p^2}$ mult./squr. based on the Karatsuba method
$\mathbb{F}_p$ Mult./Squr.
$\mathbb{F}_p$ Add./Sub.
$\mathbb{F}_{p^2}$ Mult./Squr.
$\mathbb{F}_{p^2}$ Add./Sub.
Tower Extension Field Arithmetic Hierarchy [$\mathbb{F}_p$ ... $\mathbb{F}_{p^{12}}$]
Clock Cycle [α ... α+130]
Pipelined $\mathbb{F}_{p^2}$ mult./squr.
• Done_1 of MMMB ($\mathbb{F}_p$ Mult./Squr.) actives after 38 clock cycles
• Done_2 of MMMB ($\mathbb{F}_p$ Mult./Squr.) actives after 43 clock cycles
$\mathbb{F}_{p^2}$ Arithmetic
Up to 11 pipelined $\mathbb{F}_{p^2}$ add./sub. during each $\mathbb{F}_{p^2}$ mult./squr.
$\mathbb{F}_{p^4}$ Arithmetic
$\mathbb{F}_{p^6}$ Arithmetic
$\mathbb{F}_{p^{12}}$ Arithmetic
Timing principles of field arithmetic:
• α: Start point of the diagram; and start of the first and second MMMBs
• α + 5: Start of the third MMMB unit operation
• α + 38: Done_1 of the first and second MMMBs and start of them
• α + 43: Done_2 of the first and second MMMBs; Done_1 of the third MMMB and start of that for the next operation
• α + 48: Done_2 of the third MMMB
• α + 54: End of the first $\mathbb{F}_{p^2}$ mult./squr. operation
• MASBs start and work in parallel and pipeline during the MMMB's multiplication operations

Fig. 5. Timing diagram of the tower extension field arithmetic operations (i.e., $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^4}$, $\mathbb{F}_{p^6}$, and $\mathbb{F}_{p^{12}}$ arithmetic).

plementations of $\mathbb{F}_p$ and $\mathbb{F}_{p^2}$ arithmetic discussed above. The second level consists of elliptic curve doubling/addition steps, line evaluation functions, Frobenius operators, and $\mathbb{F}_{p^{12}}$ arithmetic operations, which utilize the optimized and interleaved algorithms as well as tower extension field arithmetic hierarchy. Finally, the third level controls the high-level operations of Alg. 1, which are Miller loop (lines 2–5), Frobenius operators and final addition steps (lines 6–8), and final exponentiation (line 9). They are efficiently realized on the HW/SW codesign by using the algorithms from [22].

In the final exponentiation, an $\mathbb{F}_{p^{12}}$ inversion is required. Thanks to the tower extension field arithmetic, it can be decomposed into several additions/subtractions, multiplications, squarings, and a single inversion in $\mathbb{F}_p$, which dominates the computational cost. We compute the inversion in $\mathbb{F}_p$ with Fermat's Little Theorem that gives $a^{-1} \equiv a^{p-2} \mod p$. We compute this exponentiation using the right-to-left square-and-multiply algorithm because it allows computing multiplications in parallel with squarings and results in more efficient implementation. Because the cost of multiplications is hidden, an inversion costs only $\lfloor \log_2 p - 1 \rfloor$ squarings.

All operations are implemented as different full and segment microcode packs. The entire implementation of Alg. 1 contains 8 full and 24 segment packs. Miller loop, Frobenius operators and final addition steps, and final exponentiation consist of 1/12, 1/2, and 6/10 full/segment packs, respectively. Alg. 1 is executed in the HW/SW codesign so that, first, all microcode packs are stored in the SW side memory (i.e., DDR3). Then, for executing a specific computation step, the related full pack is loaded directly to IMEM. Then, the related segment packs are stored into the Auxiliary RAM block of the external interface unit (which can hold up to 16 segment packs). Whenever these segment packs are needed, they are loaded to the IMEM by the external interface unit. Loading of the full packs is done by the SW side, but the segment packs are loaded internally by the HW side without interaction with the SW side (see Fig. 4). The details about latencies of full and segment microcode packs transfers are reported and analyzed in the next section. It should be noted that the latencies of preparing and sending/receiving each command/status in the SW and HW sides are 45 and 5 clock cycles, respectively, which are small compared to the computation latencies in the PCP core. Furthermore, after initializing the PCP core, we need 26 commands and status transfers in total for an optimal ate pairing computation. The required clock cycles for transferring the full and segment microcode packs, commands, and statuses between the SW and HW sides are included in the reported times in the next section (i.e., Section IV).

Alg. 1 contains 8 parts (and as many full packs). Miller loop (lines 2–5) and lines 6–8 are the two first parts. The final exponentiation (line 9) divides into 6 consecutive parts, which are computed following the procedure in [39] and the specific algorithms from [22]. Alg. 1 over the BN_{126} curve with the parameters from [22] needs, in total, 14300 multiplications/squarings in $\mathbb{F}_p$ and only one inversion in $\mathbb{F}_p$.

IV. Results and Analysis

A. Implementation Setup and Results

To evaluate the performance of the HW/SW codesign, we implemented it on real hardware using Avnet Zedboard with a low-cost Xilinx Zynq-7020. The target chip includes a dual-core ARM Cortex A9 and an Artix-7 FPGA. For the SW side, we used C++ and Xilinx software development kit for developing software for a real-time operating system (RTOS). For the HW side, we used Verilog (HDL) and Vivado for implementing the design to the FPGA. The resource requirements

TABLE I
RESOURCE USAGE OF THE HW SIDE IN ZYNQ-7020 SOC PROTOTYPE

Component	LUTs	REGs	SLICEs	DSPs	BRAMs
PCP IP-Core					
PCP core	8516 (16.0%)	9641 (9.1%)	3178 (23.9%)	36 (16.4%)	17 (12.1%)
MASB	460	474	180	0	0
MMMB	1941	2292	822	12	0
Ext. Interface	68	45	21	0	1
Xilinx IP-Cores					
Mem. Intercon.	1205	1329	453	0	0
Prph. Intercon.	537	694	249	0	0
AXI DMA	2118	3490	915	0	3
AXI GPIOs	110	308	97	0	0
Processor reset	17	18	11	0	0
Total Usage	12571 (23.6%)	15480 (14.5%)	4924 (37.0%)	36 (16.4%)	21 (15.0%)

are summarized in Table I. The maximum clock frequencies for the FPGA and ARM are 105 and 667 MHz, respectively. Based on Vivado, the total power consumption of the chip is about 1.9W. All results are final post-place&route results and validated with real hardware, unless mentioned otherwise. Table II gives the number of clock cycles to compute different parts of Alg. 1 over the BN_{126} curve from [22] in the FPGA.

To demonstrate the generality of our HW/SW codesign and its efficiency on a modern programmable SoC, we implemented it on a Xilinx Zynq UltraScale+ MPSoC ZU9EG chip featuring a quad-core ARM Cortex-A53 processor running up to 1.5GHz in the SW side and a 16nm FinFET+ based FPGA in the HW side. Such a programmable SoC platform allows a significantly more powerful instance of our HW/SW codesign to be implemented in a single chip. Furthermore, to enable fair comparisons with other pairing designs, we implemented the HW side of the SoC architecture on a Xilinx Virtex-6 FPGA device. Table III reports the performance characteristics of implementation in the three above mentioned platforms. It should be noted that the results are for optimal ate pairing implementation over BN_{126} curve.

B. Related Work, Comparison, and Discussion

In this section, we consider three main categories of the pairings implementations including FPGA (and SoC), ASIC, and pure SW implementations. Also, there are many FPGA, ASIC, and SW implementations of different pairing algorithms over various curves and parameters but, for the sake of brevity, we cite only those for optimal ate pairings over BN curves that are relevant for comparisons with our implementations. Table IV shows a comparative analysis of hardware and software results of optimal ate pairings over BN curves with 126–128-bit security levels.

In the first part of Table IV for the FPGA implementations, our Virtex-6 FPGA design, which occupies only 3072 slices, 36 DSPs, and 18 BRAMs, compares favorably to other designs in respect to flexibility, scalability, programmability, and area and still offers comparable speed. Furthermore, only [36] and [41] are flexible but they are considerably slower. Flexibility is very important for HW implementations because otherwise a different HW component is needed for different pairings and parameters.

In the second part of Table IV, [28] and [26] have focused on the inflexible and customized ASIC implementations of pairing cryptographic processors. Also, [28] achieved high-speed pairing implementation in the expense of using a large area footprint. In addition, [30] proposes designing of a flexible and programmable ASIP for cryptographic pairings over BN curves in the ASIC platform but it is considerably slower. In order to compare flexibility and programmability of our HW/SW codesign with the ASIP implementation approach, we mention that our design is fundamentally a HW/SW codesign because we need the SW side, e.g., for scheduling data and packs transfers from the main memory. This allowed us to use a compact PCP core in the HW side but also provides more flexibility than ASIP approach. Of course, it would be possible to remove the SW side by extending the existing PCP core with more complex control logic (e.g., for scheduling data transfers from the main memory) similarly to an ASIP-like approach, but this would make the PCP more complex and less flexible/programmable.

Finally, in the third part of Table IV, there are SW implementations of pairings with higher speed than our flexible and programmable Hardware (HW) (i.e., FPGA) design. SW is also always flexible and programmable by nature. However, HW has many advantages besides pure speed: such as energy/op and price/core. E.g., as reported before, based on Vivado, our Zynq-7020 design consumes 1.9W whereas high-performance CPUs consume a few hundreds of Watts under full load. Comparing our PCP core and high-performance CPUs is not entirely fair because the PCP is optimized for speed/area. Besides, a single FPGA (even Zynq-7020) can have parallel PCPs to further increase the speed. Moreover, HW implementation (e.g. FPGA) is also usually easier to protect against side-channel attacks than pure SW implementation. Also, in the three parts of Table IV, there are some works that are faster but not flexible/programmable.

C. Computational Costs of Different Pairing Algorithms

It is common and effective to analyze computational costs of pairing algorithms by expressing the costs of different parts of algorithms with the numbers of $\mathbb{F}_p$ and/or $\mathbb{F}_{p^2}$ arithmetic operations. As explained before, the costs of additions/subtractions are hidden in our datapath. Furthermore, $\mathbb{F}_{p^2}$ multiplication and squaring have the same cost due to the pipeline scheme (see Fig. 5). Hence, we can estimate the costs of different steps using only the numbers of $\mathbb{F}_{p^2}$ multiplications/squarings. Because each $\mathbb{F}_{p^2}$ multiplication/squaring contains three parallel $\mathbb{F}_p$ multiplications/squarings (in our datapath) and the design computes an $\mathbb{F}_{p^2}$ multiplication/squaring with a latency $L_M = 38$, we estimate that the total number of clock cycles of a pairing algorithm in our HW/SW codesign is as follows:

$$T = \lceil C \times (\lceil M_p / 3 \rceil \times L_M + I_p \times L_I) \rceil, \tag{4}$$

TABLE II

CYCLE COUNTS OF THE HW SIDE (FPGA) FOR DIFFERENT STEPS OF OPTIMAL ATE PAIRING ALGORITHM (ALG. 1 ON BN_{126} CURVE)

	SW/HW	Miller Loop (Lines 2–5), and Final Add. (Lines 7–8)					Line 6	Final Exponentiation (Line 9)				
	IMEM LD-S/F[†]	Sqr: $f^2 \in \mathbb{F}_{p^{12}}$	$2T$, $l_{T,T}(P)$	$T+Q$, $l_{T,Q}(P)$	Mult: $f \cdot l_{T,X}(P)$	Miller-Loop[‡]	$\pi_p(Q)$, $\pi_{p^2}(Q)$	$a^{-1} \in \mathbb{F}_p / \mathbb{F}_{p^{12}}$	$f^t \in \mathbb{F}_{p^{12}} / \mathbb{G}_{\phi_6}(\mathbb{F}_{p^2})$	Frob: $f^{p^i} \in \mathbb{F}_{p^{12}}$	$f \cdot g \in \mathbb{F}_{p^{12}}$	Final Exp.[‡]
Cycle Count	46 / 1,344	475	502	619	513	101,286	86	11,938 / 13,511	34,599 / 22,707	160	703	103,064

[†] Loading a Segment or Full microcode pack (2.25 or 72 Kbit) into IMEM (IMEM LD-S/F).
[‡] Cycle counts of IMEM LD-S/F operations are considered.

TABLE III

PERFORMANCE CHARACTERISTICS OF THE PROGRAMMABLE HW/SW CODESIGN SYSTEM IN DIFFERENT PLATFORMS FOR PAIRING IMPLEMENTATION

Design Platform (SoC / FPGA Device)	SW $\leftrightarrow$ HW Interface	# of IMEM S/F Packs[†]	PCP Core in HW Side (FPGA Side)					SW Side (ARM Core)		Time[†] (ms)
			Freq.$_{max}$	Slices	DSPs	BRAMs	# Cycles[†]	Freq.$_{max}$	# Cycles[†]	
Zynq-7000 SoC (xc7z020clg484-1)	64 bit / AXI HP0	S: 24 / F: 8	105 MHz	3199 Slices[‡] (24.0%)	36 (16.4%)	18 (12.8%)	208,146	667 MHz	132,066	2.180
Virtex-6 FPGA (xc6vlx240tff1759-3)	64 bit / HW Config	S: 24 / F: 8	156 MHz	3072 Slices[‡] (8.2%)	36 (4.7%)	18 (4.3%)	208,146	—	—	1.334[*]
Zynq UltraScale+ SoC (xczu9eg-ffvb1156-2-e)	128 bit / AXI HP0	S: 24 / F: 8	230 MHz	1873 CLBs[§] (5.5%)	36 (1.4%)	18 (2.0%)	202,098	1,200 MHz	87,465	0.952

[†] For implementing optimal ate pairing over BN_{126} curve. [‡] Each slice consists of four LUTs and eight flip-flops. [§] Each CLB contains one slice and each slice provides eight LUTs and sixteen flip-flops. [*] Total computation time of the PCP core in the HW (FPGA) side without SW side time overhead.

TABLE IV

PERFORMANCE COMPARISON OF HW AND SW IMPLEMENTATIONS OF OPTIMAL ATE PAIRING OVER BN CURVES (126-128 BIT SECURITY)

Ref.	Platform	Flex	Area (Slice / DSP / BRAM)	F_{max} (MHz)	Cycle $\times 10^3$	Time (ms)
Ours	Virtex-6	Yes	3072 / 36 / 18	156	208	1.33
[37]	Virtex-6	No	5163 / 144 / 21	166	62	0.38
[25]	Virtex-6	No	7032 / 32 / 22	250	143	0.57
[27]	Virtex-6	No	4014 / 42 / 5	210	245	1.17
[36]	Virtex-4	Yes	52K / – / –	50	821	16.4
[32]	Virtex-6	No	9476 / – / –	145	80	0.56
[24]	Virtex-6	No	17560 / 11 / –	225	407	1.80
[42]	Virtex-6	No	45K / 128 / –	103	395	3.83
[34]	Virtex-6	No	7032 / 32 / 48	250	166	0.66
[35]	Virtex-6	No	5237 / 64 / 41	210	78	0.34
[33]	Virtex-6	No	5570 / 30 / –	225	80	0.35
[29]	Virtex-6	No	4380 / 131 / –	125	283	2.26
[41]	Zynq 7020	Yes	598 / – / –	324	—	134
[28]	ASIC	No	323K Gates	633	330	0.52
[26]	ASIC	No	183K Gates	204	593	2.91
[30]	ASIC	Yes	97K Gates	338	5,340	15.8
[22]	Core i7	Yes	—	2,800	2,330	0.83
[19]	Phenom II	Yes	—	3,000	1,562	0.52
[21]	Opteron II	Yes	—	2,400	1,500	0.62
	ARM A15	Yes	—	1,700	6,089	3.58

where M_p and I_p are the numbers of multiplications and inversions in $\mathbb{F}_p$, L_I is the latency of an inversion in $\mathbb{F}_p$, and C is the overhead of loading microcode packs. Based on our experiments, $C \approx 1.1$. For the optimal ate pairing from [22] considered in this paper, we have $M_p = 14300$, $I_p = 1$, and $L_I = 11938$ and (4) gives $T = 212393$. The measurements from real hardware show that real number is 208146 clock cycles and, hence, the estimate given by (4) has an error of about 2 %. Table V collects estimates of computational costs of different pairing algorithms in our HW/SW codesign by using (4) and the $\mathbb{F}_p$ operation counts available in [30].

TABLE V

COMPUTATIONAL COSTS AND CALCULATION TIME OF VARIOUS PAIRING ALGORITHMS ON BN CURVES (FROM [30]) AND OVER BN_{128} CURVE

Pairing over BN_{128} curve	# Mult. $\in \mathbb{F}_p$	# Add/Sub. $\in \mathbb{F}_p$	# Inv. $\in \mathbb{F}_p$	# FPGA Cycles	Time[†] (ms)
Opt. ate pairing	17,913	84,956	3	288,984	1.852
Ate pairing	25,870	121,168	2	386,747	2.479
η pairing	32,155	142,772	2	474,318	3.040
Tate pairing	39,764	174,974	2	580,323	3.720
Compressed η	75,568	155,234	0	1,052,942	6.750
Compressed Tate	94,693	193,496	0	1,319,417	8.458

[†] Total computation time is estimated based on the required FPGA cycle counts in Xilinx Virtex-6 FPGA device.

V. CONCLUSIONS

We presented a new HW/SW codesign for efficient computation of cryptographic pairing algorithms. It combines compact size, programmability, scalability, and flexibility (support for various pairing algorithms and curves) with relatively high performance. The architecture is optimized for the resources of modern reprogrammable SoCs such as DSPs, BlockRAMs, and hard ARM cores. In addition, the proposed SoC architecture supports microcode updates that can be used for supporting different cryptographic pairing algorithms, curves, and other parameters with the same accelerator architecture. We evaluated the proposed HW/SW codesign with real hardware and showed that the architecture can support different pairings via microcode updates and can be implemented on other reprogrammable platforms. We also investigated its efficiency in a high-end Xilinx UltraScale+ programmable SoC platform.

ACKNOWLEDGMENT

This work has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 780108 (FENTEC).

References

[1] A. J. Menezes, T. Okamoto, and S. A. Vanstone, “Reducing elliptic curve logarithms to logarithms in a finite field,” *IEEE Transactions on information Theory*, vol. 39, no. 5, pp. 1639–1646, 1993.

[2] A. Joux, “A one round protocol for tripartite Diffie–Hellman,” *Journal of Cryptology*, vol. 17, no. 4, pp. 263–276, 2004.

[3] D. Boneh and M. Franklin, “Identity-based encryption from the Weil pairing,” in *Advances in Cryptology — CRYPTO 2001*, ser. LNCS, vol. 2139. Springer, 2001, pp. 213–229.

[4] D. Boneh, B. Lynn, and H. Shacham, “Short signatures from the Weil pairing,” in *Advances in Cryptology — ASIACRYPT 2001*, ser. LNCS, vol. 2248. Springer, 2001, pp. 514–532.

[5] V. Goyal, O. Pandey, A. Sahai, and B. Waters, “Attribute-based encryption for fine-grained access control of encrypted data,” in *Proceedings of the 13th ACM Conference on Computer and Communications Security (CCS’06)*. ACM, 2006, pp. 89–98.

[6] J. Bethencourt, A. Sahai, and B. Waters, “Ciphertext-policy attribute-based encryption,” in *Proceedings of the 2007 IEEE Symposium on Security and Privacy (S&P’07)*. IEEE, 2007, pp. 321–334.

[7] M. Abdalla, M. Bellare, D. Catalano, E. Kiltz, T. Kohno, T. Lange, J. Malone-Lee, G. Neven, P. Paillier, and H. Shi, “Searchable encryption revisited: Consistency properties, relation to anonymous IBE, and extensions,” in *Advances in Cryptology — CRYPTO 2005*, ser. LNCS, vol. 3621. Springer, 2005, pp. 205–222.

[8] C. E. Z. Baltico, D. Catalano, D. Fiore, and R. Gay, “Practical functional encryption for quadratic functions with applications to predicate encryption,” in *Advances in Cryptology — CRYPTO 2017*, ser. LNCS, vol. 10401. Springer, 2017, pp. 67–98.

[9] M. Abdalla, R. Gay, M. Raykova, and H. Wee, “Multi-input inner-product functional encryption from pairings,” in *Advances in Cryptology — EUROCRYPT 2017*, ser. LNCS, vol. 10210. Springer, 2017, pp. 601–626.

[10] V. S. Miller, “The Weil pairing, and its efficient calculation,” *Journal of Cryptology*, vol. 17, no. 4, pp. 235–261, 2004.

[11] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott, “Efficient algorithms for pairing-based cryptosystems,” in *Advances in Cryptology — CRYPTO 2002*, ser. LNCS, vol. 2442. Springer, 2002, pp. 354–369.

[12] S. D. Galbraith, K. Harrison, and D. Soldera, “Implementing the Tate pairing,” in *Proceedings of Algorithmic Number Theory Symposium (ANTS 2002)*, ser. LNCS, vol. 2369. Springer, 2002, pp. 324–337.

[13] P. S. L. M. Barreto, S. D. Galbraith, C. Ó’hÉigeartaigh, and M. Scott, “Efficient pairing computation on supersingular abelian varieties,” *Designs, Codes and Cryptography*, vol. 42, no. 3, pp. 239–271, 2007.

[14] F. Hess, N. P. Smart, and F. Vercauteren, “The eta pairing revisited,” *IEEE Transactions on Information Theory*, vol. 52, no. 10, pp. 4595–4602, 2006.

[15] E. Lee, H.-S. Lee, and C.-M. Park, “Efficient and generalized pairing computation on abelian varieties,” *IEEE Transactions on Information Theory*, vol. 55, no. 4, pp. 1793–1803, 2009.

[16] F. Vercauteren, “Optimal pairings,” *IEEE Transactions on Information Theory*, vol. 56, no. 1, pp. 455–461, 2009.

[17] P. S. L. M. Barreto and M. Naehrig, “Pairing-friendly elliptic curves of prime order,” in *Selected Areas in Cryptography — SAC 2005*, ser. LNCS, vol. 3897. Springer, 2005, pp. 319–331.

[18] M. Naehrig, R. Niederhagen, and P. Schwabe, “New software speed records for cryptographic pairings,” in *Progress in Cryptology — LATINCRYPT 2010*, ser. LNCS, vol. 6212. Springer, 2010, pp. 109–123.

[19] D. F. Aranha, K. Karabina, P. Longa, C. H. Gebotys, and J. López, “Faster explicit formulas for computing pairings over ordinary curves,” in *Advances in Cryptology — EUROCRYPT 2011*, ser. LNCS, vol. 6632. Springer, 2011, pp. 48–68.

[20] D. F. Aranha, P. S. Barreto, P. Longa, and J. E. Ricardini, “The realm of the pairings,” in *Selected Areas in Cryptography — SAC 2013*, ser. LNCS, vol. 8282. Springer, 2013, pp. 3–25.

[21] R. Azarderakhsh, D. Fishbein, G. Grewal, S. Hu, D. Jao, P. Longa, and R. Verma, “Fast software implementations of bilinear pairings,” *IEEE Transactions on Dependable and Secure Computing*, vol. 14, no. 6, pp. 605–619, 2015.

[22] J.-L. Beuchat, J. E. González-Díaz, S. Mitsunari, E. Okamoto, F. Rodríguez-Henríquez, and T. Teruya, “High-speed software implementation of the optimal ate pairing over Barreto–Naehrig curves,” in *Pairing-Based Cryptography — Pairing 2010*, ser. LNCS, vol. 6487. Springer, 2010, pp. 21–39, full version: https://eprint.iacr.org/2010/354.

[23] H. Awano and M. Ikeda, “Asic coprocessor for 254-bit prime-field pairing based on general purpose arithmetic unit on quadratic extension field,” in *2018 International Conference on Advanced Technologies for Communications (ATC)*. IEEE, 2018, pp. 387–392.

[24] R. Brinci, W. Khmiri, M. Mbarek, A. B. Rabâa, and A. Bouallègue, “Efficient hardware design for computing pairings using few FPGA in-built DSPs,” 2015, https://eprint.iacr.org/2015/116.

[25] R. C. Cheung, S. Duquesne, J. Fan, N. Guillermin, I. Verbauwhede, and G. X. Yao, “FPGA implementation of pairings using residue number system and lazy reduction,” in *Cryptographic Hardware and Embedded Systems — CHES 2011*, ser. LNCS, vol. 6917. Springer, 2011, pp. 421–441.

[26] J. Fan, F. Vercauteren, and I. Verbauwhede, “Faster $\mathbb{F}_p$-arithmetic for cryptographic pairings on Barreto-Naehrig curves,” in *Cryptographic Hardware and Embedded Systems — CHES 2009*, ser. LNCS, vol. 5747. Springer, 2009, pp. 240–253.

[27] ——, “Efficient hardware implementation of fp-arithmetic for pairing-friendly curves,” *IEEE Transactions on Computers*, vol. 61, no. 5, pp. 676–685, 2011.

[28] J. Han, Y. Li, Z. Yu, and X. Zeng, “A 65 nm cryptographic processor for high speed pairing computation,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 23, no. 4, pp. 692–701, 2014.

[29] Z. Hao, W. Guo, J. Wei, and D. Sun, “Dual processing engine architecture to speed up optimal ate pairing on FPGA platform,” in *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 2016, pp. 584–589.

[30] D. Kammler, D. Zhang, P. Schwabe, H. Scharwaechter, M. Langenberg, D. Auras, G. Ascheid, and R. Mathar, “Designing an ASIP for cryptographic pairings over Barreto-Naehrig curves,” in *Cryptographic Hardware and Embedded Systems — CHES 2009*, ser. LNCS, vol. 5747. Springer, 2009, pp. 254–271.

[31] J. Sakamoto, Y. Nagahama, D. Fujimoto, Y. Okuaki, and T. Matsumoto, “Low-latency pairing processor architecture using fully-unrolled quotient pipelining montgomery multiplier,” in *2019 Asian Hardware Oriented Security and Trust Symposium (AsianHOST)*. IEEE, 2019, pp. 1–6.

[32] A. Sghaier, L. Ghammam, M. Zeghid, S. Duquesne, and M. Machhout, “Area-efficient hardware implementation of the optimal ate pairing over BN curves,” 2015, https://eprint.iacr.org/2015/1100.

[33] 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.

[34] G. X. Yao, J. Fan, R. C. Cheung, and I. Verbauwhede, “A high speed pairing coprocessor using RNS and lazy reduction,” 2011, https://eprint.iacr.org/2011/258.

[35] ——, “Faster pairing coprocessor architecture,” in *Pairing-Based Cryptography — Pairing 2012*, ser. LNCS, vol. 7708. Springer, 2012, pp. 160–176.

[36] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, “High speed flexible pairing cryptoprocessor on FPGA platform,” in *Pairing-Based Cryptography — Pairing 2010*, ser. LNCS, vol. 6487. Springer, 2010, pp. 450–466.

[37] S. Ghosh, I. Verbauwhede, and D. Roychowdhury, “Core based architecture to speed up optimal ate pairing on fpga platform,” in *International Conference on Pairing-Based Cryptography*, ser. LNCS, vol. 7708. Springer, 2012, pp. 141–159.

[38] C. Arene, T. Lange, M. Naehrig, and C. Ritzenthaler, “Faster computation of the Tate pairing,” *Journal of Number Theory*, vol. 131, no. 5, pp. 842–857, 2011.

[39] M. Scott, N. Benger, M. Charlemagne, L. J. Dominguez Perez, and E. J. Kachisa, “On the final exponentiation for calculating pairings on ordinary elliptic curves,” in *Pairing-Based Cryptography — Pairing 2009*, ser. LNCS, vol. 5671. Springer, 2009, pp. 78–88.

[40] H. Orup, “Simplifying quotient determination in high-radix modular multiplication,” in *Proceedings of the 12th Symposium on Computer Arithmetic (ARITH)*. IEEE, 1995, pp. 193–199.

[41] A. Salman, W. Diehl, and J.-P. Kaps, “A light-weight hardware/software co-design for pairing-based cryptography with low power and energy consumption,” in *2017 International Conference on Field Programmable Technology (ICFPT)*. IEEE, 2017, pp. 235–238.

[42] A. T. Wang, B. W. Guo, and C. J. Wei, “Highly-parallel hardware implementation of optimal ate pairing over Barreto-Naehrig curves,” *Integration*, vol. 64, pp. 13–21, 2019.