A FPGA Coprocessor for the Cryptographic Tate Pairing over $\mathbb{F}_p$

Alessandro Barenghi
Politecnico di Milano
Milano, Italy
barenghi@elet.polimi.it

Guido Bertoni
ST Microelectronics
Agrate Brianza, Italy
guido.bertoni@st.com

Luca Breveglieri
Politecnico di Milano
Milano, Italy
brevegli@elet.polimi.it

Gerardo Pelosi[*]
Università degli Studi di Bergamo
Dalmine, Italy
gerardo.pelosi@unibg.it

Abstract

Identity based cryptography offers a number of functional advantages over traditional public key cryptosystems and has attracted much research interest in the last few years. The computational costs demanded for such functionalities result to be significantly greater than those bounded to other methods. The overall efficiency of identity based protocols and applications is dominated by the computation of the main used primitive, namely the Tate pairing. The paper focuses on the design of a parallel hardware accelerator for the computation of the Tate pairing that makes use of arithmetics over finite fields with a large prime characteristic. Performance measurements are discussed and compared with previous solutions based on different definitions and algorithms.

Keywords: Tate pairing, identity-based cryptography, hardware accelerator, FPGA

1 Introduction

In recent years, the concept of pairings in number theory and algebraic geometry has arisen a significant research interest due to their attractive mathematical properties. Pairings have been successfully applied in the context of elliptic curve cryptography [9, 10, 13] and made possible the development of a number of new applications and protocols in the context of the identity based cryptography [13]. Identity-based cryptosystems are based on the idea that the public key itself is generated from some publicly identifiable information, such as a person's e-mail address. In such a way, before starting a transaction any user does not need to retrieve the certified copy of the recipient's public key from a public repository.

The practical realization of all the identity based protocols relies on the efficient implementation of pairings, and many research efforts have been made for the algorithmic optimization of the operational description of the pairing [4, 6, 16, 22, 26]. Nevertheless, the computation of pairing remains a relatively complicated and computationally intensive operation. Thus, dedicated hardware accelerators provide an ideal platform for pairing computation since, if scheduled carefully, many of the most intensive operations can be performed in parallel. A FPGA platform represents a good choice with respect to traditional VLSI devices thanks to its reconfigurable flexibility, rapid prototyping, reduced development costs and shortened time-to-market.

Publicly available results about hardware implementations of pairings are limited to designs that make use of finite fields with characteristic two or three [8, 18]. In the wake of results about the generation of general elliptic curves over fields of large prime characteristic [5, 14], the choice followed in the current paper of implementing a pairing using a large prime arithmetic, becomes more than appropriate and suitable to support also other public key schemes like the standard Elliptic Curve Digital Signature Algorithm (ECDSA) or the Diffie Hellman protocol variants.

The rest of this paper is organized as follows. Section 2 places mathematical preliminaries and algorithms used for the pairing computation. Section 3 outlines the design of the employed arithmetic functional units. Section 4 describes the followed design methodology and Section 5 provides the experimental evaluation of the proposed coprocessor design. Finally, Section 6 draws the conclusions and points towards future research directions.

2 Mathematical Preliminaries

In this Section we give an overview of the mathematics that underpin the definition of the Tate pairing primitive over

[*]This work was carried out under partial financial support of the Italian MiUR (Project PRIN 2006 ID-2006099978) and in part by project FSE ID-413174.

elliptic curves defined over finite fields with prime characteristic. Let $\mathbb{F}_p$ denote the finite field built over the equivalence classes generated by $\mathbb{Z}$ modulo p, where p is a prime number. Assuming $p \neq \{2, 3\}$ we can denote $E(\mathbb{F}_p)$ as the elliptic curve represented by the Weierstrass equation:

$$y^2 = x^3 + ax + b \qquad 4a^3 + 27b^2 \neq 0 \qquad a, b \in \mathbb{F}_p \quad (1)$$

The internal commutative composition law on the set of elliptic curve points is given by the well known secant and tangent rule [29]. The r-th *torsion subgroup*, $E(\mathbb{F}_p)[r]$, is the set formed by all the points of $E(\mathbb{F}_p)$ with order equal to r. In order to describe the property of such groups let n be the number of points over $E(\mathbb{F}_p)$. Assume $r \mid n \wedge r^2 \nmid n$, and consider k (from now on called *embedding degree*) to be the minimum field extension such that $r \mid p^k - 1$. An important theorem by Balasubramanian and Koblitz [2] states that, as long as $r \nmid p-1$, the r-th torsion groups are properly contained in $E(\mathbb{F}_{p^k})$ if and only if $r \mid p^k - 1$.

From a functional point of view, a pairing can be seen as a bivariate function which maps pairs of elements of an additive group $\mathbb{G}_1$ to an element of a multiplicative group $\mathbb{G}_2$ with the same order:

$$\mathbb{G}_1 \times \mathbb{G}_1 \mapsto \mathbb{G}_2$$

The main property of this mapping is somewhat similar to a multiplicative law which must be efficiently computable, distributive with respect to the sum of $\mathbb{G}_1$ as in equation 2, and such as the DLP over both groups is computationally hard. Let P, Q, P_1, P_2, Q_1, $Q_2 \in \mathbb{G}_1$:

$$\begin{aligned} e(P_1 + P_2, Q) &= e(P_1, Q)\, e(P_2, Q) \\ e(P, Q_1 + Q_2) &= e(P, q_1)\, e(P, Q_2) \end{aligned} \qquad (2)$$

Besides the difficulty of the DLP, the main security assumption about pairing functions is known as bilinear Diffie-Hellman problem (BDHP) [10]. The only groups over which it is possible to define such pairing, while granting computationally hard DLP and BDHP, are the set of points of an elliptic curve, thus assuming $\mathbb{G}_1 = E(\mathbb{F}_{p^k})$, and the elements of the multiplicative group of the underlying field, therefore assuming $\mathbb{G}_1 = \mathbb{F}^*_{p^k}$.

For a utter formal explanation of the construction of the pairing over points of elliptic curves, the reader is referred to [9, 29]. As far as the objectives of the current paper are concerned, let P and Q be two torsion points such that $P \in E(\mathbb{F}_p)[r]$, $Q \in E(\mathbb{F}_{p^k})[r]$ and $\{\mu_r\}$ be the group of the r-th roots of unity over $\mathbb{F}^*_{p^k}$, then the cryptographical Tate pairing is defined as:

$$\begin{aligned} e:\ & E(\mathbb{F}_p)[r] \times E(\mathbb{F}_{p^k})[r] \mapsto \{\mu_r\} \\ & e(P, Q) = f_P(\mathcal{D}_Q)^{\frac{p^k-1}{r}} \end{aligned} \qquad (3)$$

where $f_P(\mathcal{D}_Q)$ is a function depending on the coordinates of P and Q, and can be calculated with an iterative algorithm first proposed by Miller in [22]. The final exponentiation introduced in the above definition has the purpose of selecting a single representative from a set of results corresponding to equivalent pairs of elliptic curve points. The above definition is also referred in literature as reduced Tate Pairing.

2.1 Pairing Algorithm Selection

Since the Tate pairing has gained interest for cryptographical purposes, many efforts have been made in order to speed up the computation of the bilinear function, which is quite complex in itself. The basic algorithm consists of an extension of the classical double and add scheme, fitted with additional operations to specifically evaluate the pairing [22]. Subsequent refinements and optimizations were developed by [4, 6, 16, 26] through the elimination of unneeded computations such as the denominator of the updation formula and the elimination of the last round of the algorithm, thus resulting in a refined form called BKLS.

Thorough the introduction of bounds on the kind of curves and on the underlying finite fields, the regularity added to the system structure is exploited in order to further speed up the algorithm. Namely the Eta [3], η_t [3, 8] and Ate [17] algorithms are examples of this kind of optimizations aimed at simplifying and generalizing the calculus without weakening the security level of the system.

The choice of the parameter sizes involved in the implementation of the algorithm has to be done matching currently accepted security levels. A 1024-RSA binds the torsion group size and the extended field order to be $r \geq 2^{160}$ and $p^k \geq 2^{1024}$. In order to obtain roughly the same computational difficulty on both the DLP and the BDHP the tuning of the parameters should balance the two field sizes. Moreover, once chosen the desired values of p^k and r the embedding degree k should not exceed the ratio between them in order to correctly exploit the advantage of smaller operands, while taking into account the higher number of basic operations required to perform an operation in the extended field.

The choice of supersingular curves binds the embedding degree to be within $\{6, 4, 2\}$ for fields with characteristic $3, 2$ and $p > 3$, respectively. Therefore, as far as the $k > 2$ case goes, it is advisable to stick to ordinary curves over $\mathbb{F}_p$, where the extension $\mathbb{F}_{p^k}$ is a pairing friendly field [19], i.e. $p = 1\ mod\ 12$ and $k = 2^i 3^j$. This kind of field structure allows an easy building of a quadratic or cubic tower of polynomial extensions to represent the field elements, which results in significant arithmetical speedup. Effective methods of pairing friendly curve generation as the ones described in [14] are able to easily build curves respecting the aforementioned assumptions.

Literature works on building coprocessors for the calculation of pairings are, up to now, limited to Eta and η_t pairings which are specific to $\mathbb{F}_{3^m}$ and $\mathbb{F}_{2^m}$ [8, 24, 27], while there is still no publicly available implementation on $\mathbb{F}_p$.

The choice of $\mathbb{F}_p$ arithmetics is bound to the fact that there is no inner structure of the field other than the one defined by common sorting of the equivalence classes, while polynomial fields show a richer algebraic structure. A pioneeristic work by Coppersmith [12] pointed out that there is a possibility of exploiting some peculiarity of $\mathbb{F}_{2^m}$ and $\mathbb{F}_{3^m}$ in order to break the DLP in a faster way than brute force, therefore forcing all the related cryptosystems to raise the filed size with a consequent loss of efficiency.

Setting $\mathbb{F}_p$ arithmetic as the one to build the system on, narrows the algorithm choice down to two possibilities: the refined BKLS from [25] and the Ate pairing developed in [17]. In this work we report the results regarding the hardware implementation of pairing over a general elliptic curve following the BKLS algorithm as described in [25], where $k = 2$, $p \sim 2^{512}$ and r is a 160-bits Solinas prime (with hamming weight equal to three), in such a way to have a security level equivalent to the 1024-RSA one.

The operational description of the considered algorithm is reported at a high level specification in Algorithm 2.1.

Algorithm 2.1: Refined BKLS Algorithm [25].

```
   Input: $P \in E(\mathbb{F}_p)[r]$, $Q \in E(\mathbb{F}_{p^k})[r]$, $t = \lceil log_2(r) \rceil$,
          $r = (r_{t-1} \ldots r_0)_2$
   Output: $m = a + ib \in \mathbb{F}^*_{p^k}$
1  begin
2      $T \leftarrow P$; $m \leftarrow 1$; $r \leftarrow r - 1$
3      for $i = t - 1$ down-to $0$ do
4          $T \leftarrow 2T$                              // loop body w/o If statement:
5          $m \leftarrow m^2 \cdot f_{2T}(Q)$             // 18 mul.s and 24 add.s/sub.s
6          if $m_i = 1$ then
7              $T \leftarrow T + P$                      // If statement:
8              $m = m \cdot f_{T+P}(Q)$                  // 26 mul.s and 16 add.s/sub.s
9      return $m^{\frac{(p^k-1)}{r}}$
10 end
```

2.2 Final Exponentiation

Coming to the final part of the algorithm, the exponentiation to $(p^k - 1)/r$ needs an efficient way to lower as far as possible the number of multiplications involved in the computation. The most efficient method to obtain this on a plain finite field is known as Lucas laddering technique [26]. A schematic description of the operation performed in the Lucas chains is sketched by Algorithm 2.2.

Lucas sequences are a natural extension of the square and multiply algorithm. In the considered selection of parameters the computed pairing value is an element of the field $\mathbb{F}_{p^2}$. The quadratic extension field elements are represented as first degree polynomials since the following relation holds: $\mathbb{F}_{p^2} \cong \mathbb{F}_p(i)$ where $i \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$, $i^2 + 1 = 0$ and $p \equiv_4 3$. Therefore, assuming $f_P(\mathcal{D}_Q) = c + id$, $c, d \in \mathbb{F}_p$ and exploiting the fact that the pairing value has unitary norm, the final pairing value is computed as

$$f_P(\mathcal{D}_Q)^{\frac{p^2-1}{r}} = ((c+id)^{p-1})^m = ((c - id)^2)^m = (a+ib)^m$$

where $m = \frac{p+1}{r}$ and $a = c^2 - d^2$, $b = -2cd$.

Hence, the calculus of the final exponentiation is finalized as $(a + ib)^m = V_m(2a)/2 + U_m(2a)ib$, where V_m and U_m are the m-th terms of the Lucas sequence which can be built according to the Algorithm in 2.2.

Algorithm 2.2: Lucas Laddering [26].

```
   Input: $f = a + ib \in \mathbb{F}_{p^2}$, $a, b \in \mathbb{F}_p$, $i \in \mathbb{F}_{p^2} \backslash \mathbb{F}_p$,
          $i^2 = -1$ $p \equiv_4 3$; $t = \lceil \log_2(m) \rceil$,
          $m = (m_{t-1} \ldots m_0)_2$
   Output: $(a + ib)^m$
1  begin
2      $u_0 \leftarrow 0$; $u_1 \leftarrow 1$          // Each iteration requires:
3      $v_0 \leftarrow 2$; $v_1 \leftarrow 2a$         // 4 mul.s and 4 add.s/sub.s
4      for $i = t - 1$ down-to $0$ do
5          if $m_i = 1$ then
6              $v_0 \leftarrow v_0 v_1 - 2a$; $v_1 \leftarrow {v_1}^2 - 2$
7              $u_0 \leftarrow u_0 u_1 - 2a$; $u_1 \leftarrow {u_1}^2 - 2$
8          else
9              $v_1 \leftarrow v_0 v_1 - 2a$; $v_0 \leftarrow {v_0}^2 - 2$
10             $u_1 \leftarrow u_0 u_1 - 2a$; $u_0 \leftarrow {u_0}^2 - 2$
11     return $v_0/2 + u_0 bi$
12 end
```

3 Finite Field Arithmetic

The arithmetic operations that are required for the computation of the $e(P, Q)$ value are addition, subtraction and multiplication over the ground field $\mathbb{F}_p$. The elements of F_p are represented using a canonical unsigned binary coding with a convenient resolution to accommodate all the values in the range $[0, p - 1]$. To comply with the current security level requirements and the choices described in the previous section, in our implementation the prime p is a 512-bit number, while the quadratic extension field arithmetic operations are reduced to a sequence of $\mathbb{F}_p$ computations among coefficients of the polynomials that represents elements in $\mathbb{F}_{p^2}$.

An effective and efficient solution for the enforcement of integer arithmetic functional units with operands of the size of several hundred bits is compelled to split the design of the unit in order to manage smaller portions of the operands which must later be conveniently processed.

3.1 Modular Addition

A single component has been designed in order to perform both modular addition and subtraction. The functional unit manages the subtraction operation simply converting on the fly the second operand into two's complement representation. Modular reduction is performed with three 512-bit adders: the first one sums straightly the two operands, while the other two are structurally pipelined with the former and compute both the sum and the subtraction of the intermediate result with the modulus. The selection of the correct value is performed through looking at the MSBs of the three results. To cope with the synthesis of a fast 512-bit adder, the adopted solution consists of a core adder component with 64-bit operands used to iteratively compute the sum of the 8 slices of the original numbers; hence the final result is correctly rearranged through a ripple-carry strategy. The overall computation time can be summed up into $8\Delta T_1 + \Delta T_2$, where ΔT_1 is the time needed by the core adder to perform a word-wise operation while ΔT_2 refers to the delay needed by the second and third adder to end their own computation and bring out the correct result. Amidst a variety of design choices for fast adders, a Carry Look-Ahead design [21] offers the minimum latency solution through precomputing all the carries in a parallel fashion as soon as the operands are available. The CLA design allows a number of possible configurations of the carry prediction tree which exhibit different figures of merit in terms of latency and number of gates. The optimum time for a carry prediction network is fully logarithmic in the number of the bits of the operands.

Having determined, as reported in the next section, the necessity of a single adder unit, we concentrated on obtaining the fastest possible design, regardless of the area. Among all the CLA designs only two are able to achieve this timing result, namely the Kogge and Stone design [20], and the Slansky one [28]. The latter option resulted to be impractical because it requires a large fan-out for the logic gates involved in the prediction network, making hard to obtain an efficient synthesis by means of commercial FPGA design tools. The Kogge and Stone does not suffer from any fan-out or fan-in handicap at the expense of a larger area due to the increased number of gates and a different planning of the prediction rules. Still, the Kogge and Stone design remains the fastest CLA circuit as far as radix-2 implementations of adders go. Higher radices solutions are not suitable for FPGA synthesis due to the excessive wiring required.

3.2 Modular Multiplication

The most used method to compute multiplications over a finite field is the well known technique first proposed by Montgomery in [23]. Such a method enables to efficiently compute a modular multiplication without falling back on an explicit division, but using only an iterated series of additions and right-shifts. The operands of the multiplication are supposed to be elements of the finite field, previously converted into the Montgomery domain representation by computing a convenient power of two multiple of the operand. The advantage in doing so is that the results of a modular addition, subtraction or a multiplication are themselves in the Montgomery domain. When considering the computation of a large number of modular multiplications, as is the case with the Tate pairing calculation, it is more efficient to assume that all the arithmetic operations are performed in the Montgomery domain through an initial conversion of the input coordinates of Algorithm 2.1.Montgomery algorithm represents the most efficient way to compute modular multiplications, assuming we are dealing with a generic prime, which is our case.

As in the case of functional units for addition and subtraction, the size of the operands involved in the multiplication forces the design of the multiplier to be obtained by composing smaller functional blocks which process the two large factors using a smaller integer multiplication unit and a word-by-word approach. The FPGA implementation used in the current work adopts a multiplier most similar to the Coarsely Integrated Operand Scanning (CIOS) scheme reported in [11]. The VHDL code takes advantage of the integrated 18×18 ASIC multipliers present on the used FPGA platform, which is more extensively described in Section 5.

As described in the following Sections, the methodology used to design the pairing FPGA coprocessor needs an estimate of the execution time and of the silicon area consumption for each functional unit employed in the circuit. Execution time and occupied area for the Montgomery multiplier were evaluated by varying the word-length of the single precision multiplier involved in the CIOS design and synthesizing the corresponding circuit on our target platform for 512-bit input operands. The obtained results favored the 64-bit core multiplier model with respect to a 32-bit one, because of being able to perform the computation in one third of the time while using only 20% more area.

4 Processor Design

An hardware implementation of the pairing algorithm can expose a large amount of intrinsic parallelism in the execution of the computation by using replicated functional units and a careful scheduling analysis.

4.1 Methodology

The followed approach represents all the steps shown in Algorithm 2.1 in terms of the base field arithmetic operations and organizes them in a Direct Acyclic Graph (DAG) which is subsequently employed to perform various scheduling techniques. In this way an exhaustive scheduling campaign can be conducted to detect the best resource configuration trade-off between total execution time and required circuit area.

For an FPGA implementation such extraction would not have been possible by feeding the code through already available C-to-VHDL translators because their way of proceeding is unaware of the full structure of the algorithm.

Hence, a software scheduling tool has been developed to represent an algorithm as a DAG. This graph is first scheduled without any constraint on the resources in order to obtain an estimate of the maximum sensible number of functional units to employ in the exploration. For each possible combination of units, the software reschedules the algorithm with a resource-constrained approach and a list-based heuristic [15] in order to obtain the number of steps (ticks) needed and a reasonable approximation of the area taken by the circuit, making use of area figures of each functional unit fed as configuration options [7]. The final output of the tool is a tick-accurate description of the progression of the computation along with two further configuration files which contain a timeline of each of both the needed memory registers, which must be instantiated to memorize the intermediate results, taking care to maximize their reuse, and a time mapping of the operations executed by each replicated functional unit.

The VHDL code corresponding to the datapath pinpointed by the scheduling tool is automatically generated making use both of information about the employed functional units and of the memory registers.

The microcontroller needed to drive the datapath operations is also automatically generated as a fully hardwired Finite State Machine (FSM). The generator places each signal according to the tick-accurate description of the computation asserting each signal according to the latencies of each functional unit, also feeded as a configuration option in terms of number of ticks.

The overall component corresponding to the Tate pairing algorithm as depicted in Algorithm 2.1 is shown in Fig. 1. It makes use of a single datapath component and a master micro-controller (Fig. 2), which first loads the input data in a 32-bit word-wise fashion, setting up constants for Alg. 2.1, and then awakes the correct FSM corresponding to the basic block which has to be executed next. At the end of the computation the pairing value is memorized in a pair of internal registers and is serially transferred to the output lines when a signal is asserted from outside on a proper pin.

There are six different FSMs each of which is bound to a specific basic block of high level code, namely: loop body excluding the branch (lines 4–6 of Alg. 2.1, ML in Fig. 2), the branch code alone (lines 7–9 of Alg 2.1, MB in Fig. 2), pre-calculations needed to set up Algorithm 2.2 (Lpre in Fig. 2), high-bit ladder step (lines 5–7 of Alg. 2.2, Lc1 in Fig. 2), low-bit ladder step (lines 8–10 of Alg. 2.2, Lc0 in Fig. 2) and final post-processing block of Lucas ladder (line 13 of Alg. 2.2, Lpost in Fig. 2).

4.2 Exploration

The starting point for the exploration of the architectural solution space was the scheduling of the loop body of the pairing algorithm (lines 4–6 of Alg. 2.1), since both it is the most time consuming part and there are no obvious ways to schedule it optimally by hand. We decided to initially ignore the branch instruction present in the algorithm since it is generally executed only few times; in our case only once.

The values of estimated areas which were fed to the scheduling tool for the addition and multiplication functional units were obtained in terms of occupied slices, synthesizing each unit separately with the Xilinx suite ISE 9.1. In addition to the aforementioned arithmetical units, a third FU has been realized in order to perform move operations between registers. This FU, called from now on assigner, is composed by a simple register connected to both read and write buses and is able to work as a bridge. The choice of introducing a buffer was due to the possibility of pipelining the assignment operations.The corresponding latencies and area figures for the functional units designs described are shown in Table 1.

Table 1. Estimated characteristics of the functional units.

512-bit Unit	Latency [ticks]	Area [est. slices]]
Adder	9	$2,560$
Multiplier	173	$2,370$
Assigner	1	512

This resulted in a sensible estimate of the area fluctuations by defect, since the actual sizes provided by a synthesis tool are surely greater than the net sum of the slices occupied by each FU. The potential worsening in area figure is due to the exhaustion of the dedicated wiring resources of the FPGA, which leads to waste slices to route wires. As far as the simulations go the read and write buses were modelled as functional units with unitary latency. The area figures shown in Tab. 1 report an adder unit area greater than the one of the multiplier. This is due to the automatic exploitation by the synthesis tool of the ASIC multipliers present on the platform, as it was originally assumed at design time. This assumption does not bind the implementation to a specific FPGA model, since most Virtex FPGAs incorporate such kind of ASIC hardware.

The first round of simulations was targeted at finding out the optimal configurations with unbounded resources in order to generally assess the most optimistic achievable goals. The minimum execution time combinations can be sorted out as a function of the number of used modular multipliers, since this component is by far the most expensive one in terms of ticks needed for the execution. The obtained results are presented in Table 2.

A widely adopted architecture design is the one based on two read and one write bus as shown in Figure 1. Further simulation results not shown in the reported table put forth the adoption of only one adder since performance losses are negligible. Table 3 reports the performance losses w.r.t. the unbounded resource optimal solutions, when considering only two read buses, one write bus and a single adder.

It is evident that restraining the architecture to have only a single adder and three buses does not impact in a significant way on performances, except for the five multiplier case, where the loss of 19% is not negligible, though not excessive when balanced with the projected area savings

Table 2. Minimum tick schedules of the pairing loop body with unbounded resources.

Resources				Time	Area
$\times$	Read	Write	$+$	[ticks]	[est.slices]
1	1	1	1	3,130	6,621
2	2	1	3	1,575	15,463
3	2	1	2	1,078	16,102
4	6	2	3	909	22,214
5	1	1	3	762	25,581
6+	2	2	2	758	26,222

Table 3. Minimum tick schedules of the pairing loop body with 2 read buses, 1 write bus and 1 adder.

Mul	Time [tks]	Perf. loss [%]	Area [est.slices]
1	3,130	0	6,621
2	1,591	1.02	9,995
3	1,083	0.46	13,368
4	916	0.77	16,741
5	907	19.03	20,114
6	774	2.11	23,487

which also are around 20%, thus leading to a conservation of the area-time product. The most promising solutions appear to be the ones with three, four and five multipliers, as far as the area-time product goes.

In order to further weed out other solutions, the fact that the chip will be executing also the Lucas laddering algorithm must be considered. The Lucas algorithm does not benefit from having more than four multipliers, since there are no more than 4 parallel multiplications, while employing only three units saps performances more than 30%.

Finally, a preliminary post place and route synthesis of the overall coprocessor as described in the next section, done with a single micro-controller in order to explore multiple configurations, aimed to determine a reasonable estimate for the reachable clock rates. The results showed a loss of 38% in the clock rates for the model with 5 multipliers, with respect to the one with 4, resulting in worse global timings and thus pointing further to the latter solution as the most efficient one.

4.3 Structure

The chip is built around a classical two read, one write 512-bit wide buses architecture, as shown in Figure 1, since a completely interconnected architecture would need an enormous quantity of wires which is unmanageable with the currently available synthesis tools.

The datapath is built hooking the functional units and the registers to the buses through banks of tristate buffers. This solution allows the functional units to be fully detached from the buses while they are computing, thus lowering the capacitive load on the line and resulting in a quicker stabilization of the signals. The functional units, namely 4 multipliers, 1 adder and 1 assigner, also include buffers both at the inputs and outputs in order to preserve the values when disconnected from the buses. The register allocator described in the previous section calculated the number of needed 512-bit wide registers as ten. The memorization units are hooked one by one to the buses in order to be read and written independently without mutual blocking, which would have occurred in an alternate solution using a register file with a single set of read-write ports. The tristate banks are driven directly by microcontrollers which use them to correctly regulate the computation flow, while maintaining synchrony with the clock signal.

Due to the large width of the operands, to provide inputs and extract computed values from the chip, the coprocessor is equipped with load and store units which sequentially transfer input and output operands in 32-bit-wide words. The activation of the load units is triggered by the master microcontroller when the *start* signal (Fig. 1) is issued from outside the chip, while the write units are activated by a *write_results* external signal which enables the outputs to be exposed.

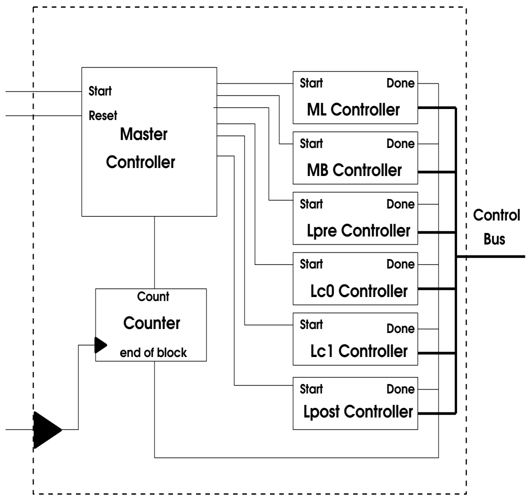

Figure 2. Master microcontroller structure.

5 Experimental Results

After fully determining the structure of the coprocessor, a well known and widely used platform has been used in order to obtain a full synthesis FPGA implementation. The final result represents the first publicly available data about a FPGA coprocessor for the Tate pairing over $\mathbb{F}_p$. The chosen development toolchain is Xilinx ISE 9.1.03i on Linux OS for amd64 architecture; the synthesizer is configured to

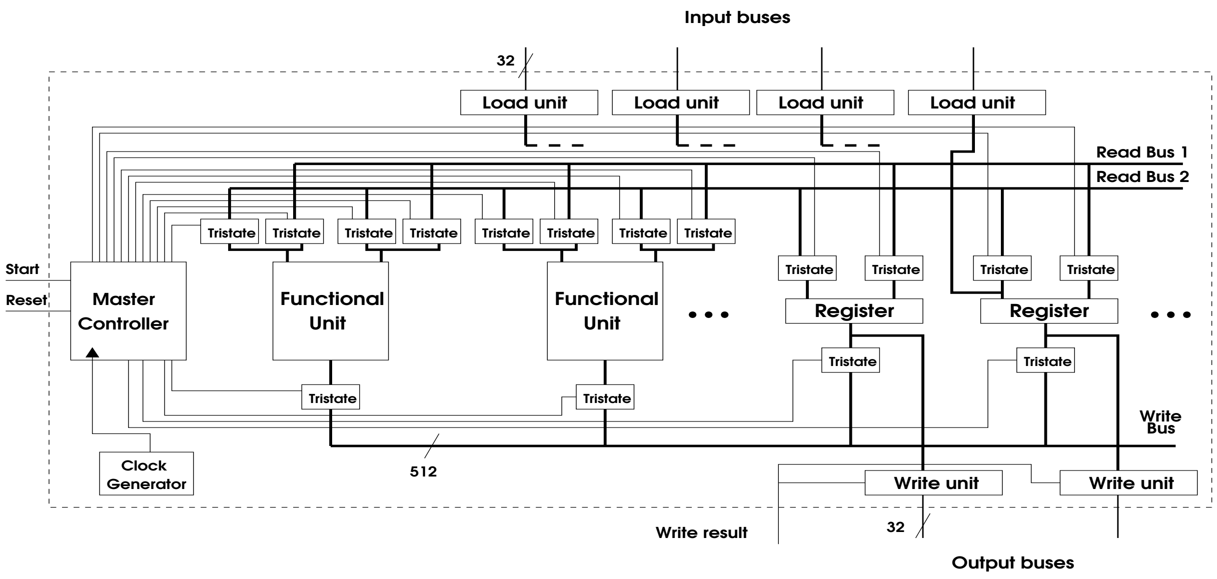

Figure 1. Structure of the complete coprocessor chip.

use area as the target figure for synthesis optimization and instructed to use a balanced criterion for mapping.

The target chip for implementation was Virtex-2 8000 (XC2V8000-5FF1152), due both to its wide diffusion and to the fact that it does not include unnecessary (at least for our purpose) ASIC hardware on the chip in contrast with the the more expensive Virtex 2 Pro line, used in other literature works. Virtex 2 8000 contains 46592 slices arranged in 112×104 4–slices CLB and 168 18×18 ASIC multiplier blocks, which are automatically used in order to implement the core of the modular multipliers.

Table 4. Pairing HW implementations.

Solution	Algorithm	k	Operand Size[bits]	MOV Level	No. of Mult.s
$\mathbb{F}_p$ (Ours)	BLKS	2	512	$1,024$	4
$\mathbb{F}_{3^m}$ [18]	DL	6	154	924	18
$\mathbb{F}_{3^m}$ [8]	η_t	6	154	924	1
$\mathbb{F}_{3^m}$ [24]	η_t	6	154	924	3

A comparison of our synthesis result with the current hardware achievements for Tate pairing calculation is summed up in Table 5. Table 4 reports a list of the system parameters adopted for each of the cited implementations. Particular care should be exercised when comparing results over different fields since the differences in arithmetics and operands size impact dramatically on the type of allowed optimizations.

The correct metric which must be used to compare implementations over different fields is the security level guaranteed for the DLP over the employed fields (referred in Table 4 as MOV Level).

Among the reported works, the one described in [8] was targeted at building a small coprocessor, also through exploiting the ASIC hardware present on the chosen Virtex 2 Pro 4 FPGA, thus resulting in a significantly compact architecture at the cost of sacrificing flexibility in the choice of the platform. The work is based on a single multipurpose unit, in order to further save area, thus resulting in a very compact design which, however, is not able to exploit the intrinsic parallelism of the architecture. An example of partial exploitation of parallelism is given by [18], where the operations over $\mathbb{F}_{3^{mk}}$ were analyzed through a handmade scheduling thus conceiving a special multiplier unit designed to work with polynomials. This yielded a particularly large area usage and cut down the maximum working frequency due to complex critical pathways. In the work reported in [24] a semi-automatized exploration of the solution space was conducted, leading to an analysis of the possible tradeoffs both oriented to throughput and area. The reported solution is the one targeted to speed, regardless of area savings.

Table 5. Post Place & Route synthesis results.

Solution	FPGA model	Area [Slices]	Freq. [MHz]	Time [ms]
$\mathbb{F}_p$ (Ours)	V2 8000	$33,857$	135	1.61
$\mathbb{F}_{3^m}$ [18]	V2Pro125	$55,616$	15	0.85
$\mathbb{F}_{3^m}$ [8]	V2Pro4	$1,888$	147	0.22
$\mathbb{F}_{3^m}$ [24]	V2Pro4	$10,000$	70	0.18

6 Concluding Remarks

The results illustrated in this paper show that it is possible to build a coprocessor for the calculation of the Tate pairing over $\mathbb{F}_p$ and reach a computation time within one order of magnitude from the ones over $\mathbb{F}_{3^{mk}}$, and therefore well in the range of usability. To our knowledge, ours is the first realistic result for the HW computation of the Tate pairing over a prime field $\mathbb{F}_p$. This is particularly interesting when considering the greater intrinsic resilience to attacks of plain fields opposed to the polynomial ones. The main issue with this coprocessor is the large occupied area, mainly due to the complex wiring required during the place and route phase, which is caused by the inner architecture of the target FPGA. In particular, the maximum size of a slice array on the chip is of 448 cells, thus forcing a significant waste of area for wiring (around 25%); this issue can be addressed both by employing a smaller number of multipliers, at the expense of performance, and by re-targeting the synthesis to an ASIC platform which can freely organize its layout. An interesting aspect is the fact that there is still enough space on the FPGA to place additional controllers which exploit the 512-bit wide $\mathbb{F}_p$ arithmetics and implement other IBE schemes or the ECDSA standard, thus obtaining far greater security levels.

References

[1] *Third International Conference on Information Technology: New Generations (ITNG 2006), 10-12 April 2006, Las Vegas, Nevada, USA*. IEEE Computer Society, 2006.

[2] R. Balasubramanian and N. Koblitz. The improbability that an elliptic curve has subexponential discrete log problem under the menezes - okamoto - vanstone algorithm. *J. Cryptology*, 11(2):141–145, 1998.

[3] P. S. Barreto, S. D. Galbraith, C. O. Héigeartaigh, and M. Scott. Efficient pairing computation on supersingular abelian varieties. *Des. Codes Cryptography*, 42(3):239–271, 2007.

[4] P. S. L. M. Barreto, H. Y. Kim, B. Lynn, and M. Scott. Efficient algorithms for pairing-based cryptosystems. In *CRYPTO '02: Proceedings of the 22nd Annual International Cryptology Conference on Advances in Cryptology*, pages 354–368, London, UK, 2002. Springer-Verlag.

[5] P. S. L. M. Barreto, B. Lynn, and M. Scott. Constructing elliptic curves with prescribed embedding degrees. Cryptology ePrint Archive, Report 2002/088, 2002. `http://eprint.iacr.org/`.

[6] P. S. L. M. Barreto, B. Lynn, and M. Scott. On the selection of pairing-friendly groups. In M. Matsui and R. J. Zuccherato, editors, *Selected Areas in Cryptography*, volume 3006 of *Lecture Notes in Computer Science*, pages 17–25. Springer, 2003.

[7] G. Bertoni, L. Breveglieri, P. Fragneto, and G. Pelosi. Parallel hardware architectures for the cryptographic tate pairing. In *ITNG* [1], pages 186–191.

[8] J.-L. Beuchat, N. Brisebarre, J. Detrey, and E. Okamoto. Arithmetic operators for pairing-based cryptography. In P. Paillier and I. Verbauwhede, editors, *CHES*, volume 4727 of *Lecture Notes in Computer Science*, pages 239–255. Springer, 2007.

[9] I. Blake, G. Seroussi, N. Smart, and J. W. S. Cassels. *Advances in Elliptic Curve Cryptography (London Mathematical Society Lecture Note Series)*. Cambridge University Press, New York, NY, USA, 2005.

[10] D. Boneh and M. K. Franklin. Identity-based encryption from the weil pairing. *SIAM J. Comput.*, 32(3):586–615, 2003.

[11] Çetin Kaya Koç, T. Acar, and J. Burton S. Kaliski. Analyzing and comparing montgomery multiplication algorithms. *IEEE Micro*, 16(3):26–33, 1996.

[12] D. Coppersmith. Fast evaluation of logarithms in fields of characteristic two. *IEEE Transactions on Information Theory*, 30(4):587–593, 1984.

[13] R. Dutta, R. Barua, and P. Sarkar. Pairing-based cryptographic protocols: A survey. Cryptology ePrint Archive, Report 2005/64, 2004.

[14] D. Freeman, M. Scott, and E. Teske. A taxonomy of pairing-friendly elliptic curves. Cryptology ePrint Archive, Report 2006/372, 2006. `http://eprint.iacr.org/`.

[15] D. D. Gajski, N. D. Dutt, A. C.-H. Wu, and S. Y.-L. Lin. *High-Level Synthesis: Introduction to Chip and System Design*. Kluwer Academic, 1992.

[16] S. D. Galbraith, K. Harrison, and D. Soldera. Implementing the tate pairing. In *ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory*, pages 324–337, London, UK, 2002. Springer-Verlag.

[17] F. Hess, N. P. Smart, and F. Vercauteren. The eta pairing revisited. *IEEE Transactions on Information Theory*, 52(10):4595–4602, 2006.

[18] T. Kerins, W. Marnane, E. M. Popovici, and P. Barreto. Efficient hardware for the tate pairing calculation in characteristic three. In *CHES: International Workshop on Cryptographic Hardware and Embedded Systems, CHES, LNCS*, October 2005.

[19] N. Koblitz and A. Menezes. Pairing-based cryptography at high security levels. In N. P. Smart, editor, *IMA Int. Conf.*, volume 3796 of *Lecture Notes in Computer Science*, pages 13–36. Springer, 2005.

[20] P. Kogge and H. Stone. A parallel algorithm for the efficient solution of a general class of recurrence equations. *IEEE Trans. Computers*, C-22:786–793, 1973.

[21] I. Koren. *Computer arithmetic algorithms*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1993.

[22] V. Miller. Short program for functions on curves, 1986.

[23] P. L. Montgomery. Modular multiplication without trial division. *Mathematics of Computation*, 44(170):519–521, 1985.

[24] R. Ronan, C. O'Eigeartaigh, C. C. Murphy, M. Scott, T. Kerins, and W. P. Marnane. An embedded processor for a pairing-based cryptosystem. In *ITNG* [1], pages 192–197.

[25] M. Scott. Computing the tate pairing. In A. Menezes, editor, *CT-RSA*, volume 3376 of *Lecture Notes in Computer Science*, pages 293–304. Springer, 2005.

[26] M. Scott and P. S. L. M. Barreto. Compressed pairings. In M. K. Franklin, editor, *CRYPTO*, volume 3152 of *Lecture Notes in Computer Science*, pages 140–156. Springer, 2004.

[27] C. Shu, S. Kwon, and K. Gaj. Fpga accelerated tate pairing based cryptosystems over binary fields. Cryptology ePrint Archive, Report 2006/179, 2006. `http://eprint.iacr.org/`.

[28] J. Slansky. Conditional sum addition logic. *IRE Trans. Electronic Computers*, 9(6):226–231, 1960.

[29] L. C. Washington. *Elliptic Curves: Number Theory and Cryptography*. CRC Press, Inc., Boca Raton, FL, USA, 2003.