Area-Efficient Hardware Implementation of the Optimal Ate Pairing over BN curves. *

Anissa Sghaier[1], Ghammam Loubna[1.2], Medien Zeghid[1.3]
Sylvain Duquesne[2] and Mohsen Machhout[1]
[1] Faculty of Sciences, University of Monastir , $E\mu E$ Lab
Monastir 5019, Tunisia, sghaieranissa@yahoo.com
[2] Rennes I University, IRMAR lab, UMR CNRS 6625, Beaulieu
Campus 35042 Rennes Cedex France, ghammam.loubna@yahoo.fr
[3] Higher Institute of Applied Sciences and Technology,
Taffala city 4003 Sousse, Tunisia

January 25, 2016

Abstract

To have an efficient asymmetric key encryption scheme such as elliptic curves, hyperelliptic curves, pairing etc., we have to go through an arithmetic optimization then a hardware one. Taking into consideration restricted environments' compromises, we should strike a balance between efficiency and memory resources. For this reason, we studied the mathematical aspect of pairing computation and gave new development of the methods that compute the hard part of the final exponentiation in [2]. They prove that these new methods save an important number of temporary variables, and they are certainly faster than the existing one. In this paper, we will also present a new way of computing Miller loop, more precisely in the doubling algorithm. So we will use this result and the arithmetic optimization presented in [2]. Then, we will apply hardware optimization to find a satisfactory design which give the best compromise between area occupation and execution time. Our hardware implementation on a Virtex-6 FPGA(XC6VHX250T) used only 5976 Slices, 30 DSP, which is less resources used compared with state-of-the-art hardware implementations, so we can say that our approach cope with the limited resources of restricted environment. **keywords**: BN curves, Optimal Ate Pairing, Arithmetic optimization, memory resources, hardware implementations.

1 Introduction

The performance of pairing based protocols depends on the efficiency of pairing computation. The computation of these pairings consists of two parts: the Miller loop and then the final exponentiation. The Miller loop consists of the computation of the function $f_{u,P}$ and then evaluate this function on the point Q, where P and Q are two points of an elliptic curve E. The function $f_{u,P}$ is defined by its divisor

*This work was supported in part by Laboratory of electronic and microelectronic of Monastir (LR99ES30) and Henri Lebesgue center in France (ANR-11-LABX-0020-01).

$\mathrm{Div}(f_{u,P}) = u(P) - ([u]P) - (u-1)(P_\infty)$ where u is an integer and P_∞ denotes the point at infinity. The computation of this function is done thanks to the equality of Miller:

$$f_{[i+j],P} = f_{[i],P} f_{[j],P} \frac{l_{[i]P,[j]P}}{v_{[i+j]P}}$$

where

- $l_{[i]P,[j]P}$ is the line passing through $[i]P$ and $[j]P$,
- $v_{[i+j]P}$ is the vertical to E at $[i+j]P$.

The efficiency of the Miller step depends certainly on the bit length of u and also on its hamming. After computing the Miller loop $f_1 = f_{u,P}(Q)$, we have to raise the result f_1 to the power $\frac{p^k-1}{r}$. Thanks to the cyclotomic polynomial, this exponent can be simplified using the following decomposition (let $k' = k/2$):

$$\frac{p^k-1}{r} = \left(p^{k'}-1\right)\left[\frac{(p^{k'}+1)}{\phi_k(p)}\right]\left[\frac{\phi_k(p)}{r}\right]$$

with r is a large prime divisor of the order of the group of rational points of E, and k is the embedding degree which is defined as the smallest integer such that r divides $p^k - 1$.
In our case the embedding degree k is equal to 12. Then we compute the final exponentiation

$$\frac{p^{12}-1}{r} = \left(p^6-1\right)\left(p^2+1\right)\frac{p^4-p^2+1}{r}$$

on two steps: at first we compute $f = f_1^{\left(p^6-1\right)\left(p^2+1\right)}$ which is the easy part, then we have to evaluate f to the power $\frac{p^4-p^2+1}{r}$ which is the hard part of the final exponentiation.

New hardware approaches are needed in order to implement some computational heavy and power consuming functions in order to meet the current restricted environment requirements. In general, hardware implementations have been proved better approaches compared with the software developments, in the terms of throughput, area and operating frequency, but every algorithm should be demonstrated in software before coming to hardware. In this paper we will be interested by the FPGA implementation of Optimal Ate Pairing. During the last years, several hardware implementations of bilinear pairings, targeting the 128-bit security level, have been presented. In 2011, Ray C.C.Cheung et al. [4] give two designs using the Residue Number System which is suitable for parallel architectures and lazy reduction to speed up optimal ate pairing at 126-bit security. In 2012, J.Fan and al. [5] present a hardware implementation of $\mathbb{F}_p$-arithmetic for pairing, and they introduce a new reduction algorithm for polynomial form modulo which is Hybrid Modular Multiplication composed of four phases, polynomial multiplication, a partial coefficient reduction, polynomial reduction and coefficient reduction.Then in 2013, S.Ghosh and al. [8] speed up optimal ate pairing computation having 126-bit security by exploiting IP cores available in modern FPGAs and they present a pipe-lined data-paths for $\mathbb{F}_p$-operations. Until now, many hardware and software implementations was presented. To speed-up pairing computation, we should speed-up arithmetic operations. For this reason, many mathematical studies are done to accelerate arithmetic calculus. Our work focuses on two important mathematical results: First, optimized algorithms presented by Duquesne et al. in [2]. Second, new way of computing Miller loop detailed in section 2.2. Our approach touch resource constrained embedded systems, which can benefit greatly from employing cryptographic algorithms that are tuned to consume as little system resources as possible, while at the same time providing reasonable performance. The latest years, pairing implementations have been very attractive for the hardware designers, and retrained environment which have limited computing power and minimized storage capacity. Therefore, to provide good level of security for these applications, we should define a flexible architecture.

In this paper, we are interested, first, by the arithmetic optimization concerning the first part optimal ate pairing algorithm, which is the Miller Loop computation, so we present a new way of computing it, more precisely in the doubling algorithm. In addition we will apply our results given in [2] and our hardware optimization to find an efficient architecture computing the optimal ate pairing, where we found a compromise between efficiency and memory resources. The proposed architecture design is based on an hybrid methodology. This paper deals with three issues, namely, proposing architecture for hardware implementation on FPGA, optimizing the architecture and comparing the performance metrics of different FPGA, that implement a pairing. Our implementation proved the results given by Duquesne et al. in [2] and verified that is more efficient than others implementations presented in the literature and that our design is the most performing in term of area and cycle number which let it more suitable for restricted environments.

The remaining paper is organized as follow, the second section is a presentation of BN curves, Optimal ate pairing and also we detailed the computation of doubling step where we present a new variant of the original work and we detailed also addition step. In section 3, the proposed system presented and the internal components of this architecture are described in detail. Hardware optimization are given in section 4. The synthesis results of the FPGA implementation and a comparison with other related works are presented in section 5. Finally, conclusions and observations are given in section 6.

Notations and Assumptions:
In the rest of this paper we use the following notations.

- M_k is a multiplication in $\mathbb{F}_{p^k}$.
- S_k is a squaring in $\mathbb{F}_{p^k}$.
- F_k is a Frobenius map application in $\mathbb{F}_{p^k}$.
- I_k is an inversion in $\mathbb{F}_{p^k}$.
- A_k is an addition in $\mathbb{F}_{p^k}$.
- w_e is the Hamming weight of an integer $\mid e \mid$.
- l_e is the length of $\mid e \mid$ in base 2.

For simplicity, we use M, A, S and I instead of M_1, A_1, S_1 and I_1.
Practically, when we compute the final exponentiation, we must perform the operations one by one in each line. For that, we assume that all operations of type $a \leftarrow ab$ are possible in place. This hypothesis is reasonable because our computations are in the

field extensions. Anyway, our results would be similar if such operations were not possible.

2 Background of pairings

In this section, we will give a brief background of pairings: BN-Curves, Miller Loop, and Final exponentiation.

2.1 BN curves presentation

Barreto and Naherig presented in [31] a method to generate pairing friendly elliptic curves over a prime field $\mathbb{F}_p$ with embedding degree $k = 12$ and a prime order n. These curves are called BN curves and are defined over $\mathbb{F}_p$ by the following equation

$$E:\ y^2 = x^3 + b,$$

where $b \neq 0$ is nor a square neither a cube and by a parameter u such that

$$\begin{array}{l} t = 6u^2 + 1 \\ n = 36u^4 + 36u^3 + 18u^2 + 6u + 1 \\ p = 36u^4 + 36u^3 + 24u^2 + 6u + 1 \end{array}$$

where t is the trace of Frobenius map on the curve. The parameter u is chosen such that E has prime order. We assume this is the case in this paper, and more precisely in our implementations we will choose a special value for u given in the following example.

Example 1 *Nogami et al. [29] have suggested the following choice of*

$$u = -(4080000000000001)_{16}.$$

The Hamming weight of $-u$ is $w_u = 3$ and the length of $-u$ in base 2 is $l_u = 63$.

Barreto-Naherig (BN) curves are the ideal solution for computing pairing for a 128 bits security level, specially for computing Optimal Ate pairing which is the following map:

$$\begin{array}{rcl} \mathbf{e_{opt}} : G_2 \times G_1 & \to & G_3 \\ (Q, P) & \longmapsto & \left(f_{s,Q}(P) l_{[s]Q,\phi_p(Q)}(P) l_{[s]Q+\phi_p(Q),-\phi_p^2(Q)}(P)\right)^{\frac{p^k-1}{r}} \end{array}$$

with:

- $s = 6u + 2$,
- ϕ_p the Frobenius map,
- $G_1 = E(\mathbb{F}_p)[r]$
- $G_2 = E'(\mathbb{F}_{p^2})[r]$
- $G_3 = \mathbb{F}_{p^{12}}^{\times}$

To compute a pairing, as we said, we have two steps: Miller loop and final exponentiation. At first, we have to compute the Miller function. Then, we carry out the result to the power $\frac{p^{12}-1}{r}$ which is the final exponentiation. Let us at first present the Miller loop in the next section.

2.2 Miller loop

In the case of Optimal Ate pairing, the Miller function consists in the computation of the following expression:

$$f = f_{s,Q}(P) l_{[s]Q,\phi_p(Q)}(P) l_{[s]Q+\phi_p(Q),-\phi_p^2(Q)}(P)$$

The Miller loop is computed thanks to Algorithm 1 [32]. Points doubling and corresponding their line evaluations dominate the cost of Miller loop. Also, the additions points with their corresponding line evaluations depend to the Hamming weight of the Miller variable u. Pairing can be computed over elliptic curves represented in any coordinates system such affine coordinates, Jacobien coordinates and projective coordinates. The choice of projective coordinates has proven especially advantageous at the 128-bit security level for single pairing computation [32] and it is our case in this paper.

Algorithm 1 : Miller loop of Optimal Ate pairing

```
**Input:** $P \in \mathbb{G}_1$, $Q \in \mathbb{G}_2$, $s = \mid 6u + 2 \mid = \sum_{i=0}^{log_2(s)} s_i 2^i$
**Output:** $e_{opt}(Q, P)$
1: $d \leftarrow l_{Q,Q}(P),\ S \leftarrow 2Q,\ e \leftarrow 1$
2: **if** $s_{\lfloor log_2(s) \rfloor - 1} = 1$ **then** $e \leftarrow l_{S,Q}(P), S \leftarrow T + Q$
3: $f \leftarrow d.e$
4: **for** $i = \lfloor log_2(s) \rfloor - 2$ **down to** $0$ **do**
5: $f \leftarrow f^2.l_{S,S}(P), S \leftarrow 2S$
6: **if** $s_i = 1$ **then** $f \leftarrow f.l_{S,Q}(P), S \leftarrow S + Q$
7: **end for**
8: If $u < 0$ $S \leftarrow -S, f \leftarrow f^{p^6}$
9: $Q_1 \leftarrow \phi_p(Q), Q_2 \leftarrow \phi_{p^2}(Q)$
10: $d \leftarrow l_{S,Q_1}(P), S \leftarrow S + Q_1, e \leftarrow l_{S,Q_2}(P), S \leftarrow S - Q_2, f \leftarrow f.(d.e)$
11: **return** $f$
```

Now, we present at first the projective coordinates. Then to perform step 5 in Miller algorithm, we present the way of the computation of $l_{Q,Q}(P)$ which is the tangent to E at the point Q and the doubling step. Also to compute step 6 in algorithm 1 we have to compute $l_{S,Q}(P)$ the line joint S and Q evaluated at P and the addition step.
The elliptic curve E which we consider in our implementation is defined over $\mathbb{F}_p$ in affine coordinates by:

$$y^2 = x^3 + 2$$

As we said, we will compute the pairing in the projective coordinates. So we have to make the following change of variables:

$$(x, y) = \left(\frac{X}{Z}, \frac{Y}{Z} \right)$$

So the elliptic curve equation in the projective coordinates E is given by:

$$E : y^2 z = x^3 + 2z^3.$$

The twist curve is so presented by:

$$E' : y^2 z = x^3 + (1 - i)z^3.$$

Computation of $l_{Q,Q}(P)$ and the doubling step The slope of the tangent at S is

$$\lambda_{S,S} = \frac{3x_S^2}{2y_S} = \frac{3x'^2_{S'}}{2y'_{S'}}\gamma = \frac{N_1}{D_1}\gamma$$

where $N_1 = 3x_{S'}^2$ and $D_1 = 2y_{S'}z_{S'}$ in $\mathbb{F}_{p^2}$. Then,

$$l_{S,S}(P) = y_P - y_S - \lambda_{S,S}(x_P - x_S) = y_P - \frac{y_{S'}}{z_{S'}}\gamma^3 - \frac{3x_{S'}^2}{2y_{S'}z_{S'}}\gamma(x_P - \frac{x_{S'}}{z_{S'}}\gamma^2) = \frac{N_2}{D_2}$$

where $D_2 = 2y_{S'}z_{S'}^2$ and

$$N_2 = y_P D_2 - 3x_P x_{S'}^2 z_{S'}\gamma + (3x_{S'}^3 - 2y_{S'}^2 z_{S'})\gamma^3.$$

Because D_1 is in $\mathbb{F}_{p^2}$ it suffices to compute in the doubling step in Miller loop $f \leftarrow f^2$ then updating f by computing $f \leftarrow fN_2$.
These operations cost in projective coordinate $S_{12} + 15M_2 + 21A_2 + 4A'_2$.

Algorithm 2:

Input: $x'_P = -3x_P, y_P$ $X_1,\ Y_1,\ Z_3$ **Output:**$X_3,\ Y_3,\ Z_3,$ $t_0,\ t_1,\ t_2$	Complexity		Complexity
1. $T_1 \leftarrow X_1^2$	S_2	16. $t_3 \leftarrow t_0 + Y_3$	A_2
2. $T_2 \leftarrow Y_1^2$	S_2	17. $Z_3 \leftarrow T_2 - t_3$	A_2
3. $T_3 \leftarrow Z_1^2$	S_2	18. $X_3 \leftarrow X_3Z_3$	M_2
4. $X_3 \leftarrow X_1 + Y_1$	A_2	19. $t_3 \leftarrow T_2 + t_3$	A_2
5. $X_3 \leftarrow X_3^2$	S_2	20. $t_3 \leftarrow t_3^2$	S_2
6. $X_3 \leftarrow X_3 - T_1$	A_2	21. $Y_3 \leftarrow Y_3^2$	S_2
7. $X_3 \leftarrow X_3 - T_2$	A_2	22. $t_3 \leftarrow t_3 - Y_3$	A_2
8. $t_1 \leftarrow Y_1 + Z_1$	A_2	23. $Y_3 \leftarrow 2Y_3$	A_2
9. $t_1 \leftarrow t_1^2$	S_2	24. $Y_3 \leftarrow t_3 - Y_3$	A_2
10. $t_1 \leftarrow t_1 - T2$	A_2	25. $Z_3 \leftarrow T_2t_1$	M_2
11. $t_1 \leftarrow t_1 - T3$	A_2	26. $Z_3 \leftarrow 2Z_3$	A_2
12. $t0 \leftarrow (1-i)T_3$	A_2	27. $Z_3 \leftarrow 2Z_3$	A_2
13. $t_3 \leftarrow 2t_0$	A_2	28. $t_3 \leftarrow T_2 - t_0$	A_2
14. $t_0 \leftarrow t_0 + t_3$	A_2	29. $t_0 \leftarrow y_Pt_1$	$2M$
15. $Y_3 \leftarrow 2t_0$	A_2	30. $t_1 \leftarrow x'_PT_1$	$2M$

Let $S = (X_1, Y_1, Z_1) \in E'(\mathbb{F}_{p^2})$ a point in projective coordinates, we compute the doubling of S so $2S$ with the following formula presented in [1]:

- $X_3 = 2x_Ty_Tz_T(9x_T^3 - 8y_T^2z_T)$
- $Y_3 = 9x_T^3(4y_T^2z_T - 3x_T^3) - 8y_T^4z_T^2$
- $Z_3 = (2y_Tz_T)^3$

To simplify these expressions, we can use the equation of the curve where we have $y_{S'}^2z_{S'} - (1-i)z_{S'}^2 = x_{S'}^3$. Then we got:

- $X_3 = 2x_Ty_Tz_T^2(y_T^2 - 9(1-i)z_T^2)$

- $Y_3 = z_T^2\left((y_T^2 + 9(1-i)z_T^2)^2 - 108(1-i)^2 z_T^4\right)$
- $Z_3 = (2y_T z_T)^3$

So that:

$$N_2 = y_P(2y_T z_T^2) - 3x_P x_T^2 z_T \gamma + \left(y_T^2 z_T - 3(1_i)z_T^3\right)\gamma^3$$

The advantage of these expressions that they are a multiple of z_T^2 which is an element of $\mathbb{F}_{p^2}$. So we can simplify x_T, y_T and z_T and get the following formulae:

- $X_3 = 2x_{S'}y_{S'}(y_{S'}^2 - 9(1-i)z_{S'}^2$
- $Y_3 = \left((y_{S'}^2 + 9(1-i)z_{S'}^2)^2 - 108(1-i)^2 z_{S'}^4\right)$
- $Z_3 = 8y_T^3 z_T$

Then;

$$N_2 = 2y_{S'}z_{S'}y_P - 3x_P x_{S'}^2\gamma + \left(y_{S'}^2 - 3(1_i)z_{S'}^2\right)\gamma^3$$

Let $X_1,\ Y_1,\ Z_3$ the projective coordinates of S' in $E'(\mathbb{F}_{p^2})$ and $X_3,\ Y_3,\ Z_3$ the projective coordinates of $2S'$. We consider that the tangent to E at S evaluated on P is $l_{S,S} = t_0 + t_1\gamma + t_3\gamma^3$. So, we present Algorithm 2, with T_1, T_2, and T_3 are the used temporary variables. This algorithm requires $2M_2 + 7S_2 + 4M + 18A_2 + A_2'$ for computing the line N' which is $l_{S,S}(P)$ and updating $S' \leftarrow 2S'$ instead of $2M_2 + 7S_2 + 4M + 23A_2 + A_2'$ if we use the Jacobien coordinates. In this algorithm also for computing $2XY$, we use the fact that $2XY = (X+Y)^2 - X^2 - Y^2$. In our case it is better to compute directly $2XY$ by a multiplication and an addition.

Computation of $l_{S,Q}(P)$ and the addition step We assume that S must be different to $\{Q, -Q\}$, The slope of the line $l_{S,Q}$ is

$$\lambda_{S,Q} = \frac{y_S - y_Q}{x_S - x_Q} = \frac{y'_{S'} - y'_{Q'}}{x'_{S'} - x'_{Q'}}\gamma = \frac{N_1'}{D_1'}\gamma$$

Where $N_1' = y_{S'} - y_{Q'}z_{S'}$ and $D_1' = x_{S'} - x_{Q'}z_{S'}$, $D_1' \in \mathbb{F}_{p^2}$.
The line $l_{S,Q}$ evaluated on the point P is:

$$\begin{aligned} l_{S,Q}(P) &= y_P - y_Q - \lambda_{S,Q}(x_P - x_Q) \\ &= y_P - \frac{x_P(y_{S'} - y_{Q'}z_{S'})}{x_{S'} - x_{Q'}z_{S'}}\gamma + \left(\frac{x_{Q'}(y_{S'} - y_{Q'}z_{S'})}{x_{S'} - x_{Q'}z_{S'}} - y_{Q'}\right)\gamma^3 \\ &= y_P - \frac{x_P N_1'}{x_{S'} - x_{Q'}z_{S'}}\gamma + \left(\frac{x_{Q'}y_{S'} - y_{Q'}x_{S'}}{x_{S'} - x_{Q'}z_{S'}}\right)\gamma^3 \\ &= \frac{N_2'}{D_2'} \end{aligned}$$

Because $D_2' \in \mathbb{F}_{p^2}$ then we will evaluate $l_{S,Q}$ as

$$l_{S,Q} = y_P D_2' - x_P(N_1')\gamma + (x_{Q'}y_{S'} - y_{Q'}x_{S'})\gamma^3.$$

Finally, to compute the addition of the two points S' and Q', we need to the following expressions:

- $C = (N_1')^2 z_{S'} + (D_1')^3 - 2(D_1')^2 x_{S'}$

- $X_3 = D'_1.C$
- $Y_3 = N'_1((D'_1)^2 x_{S'} - C) - (D'_1)^3 y_{S'}$
- $Z_3 = (D'_1)^3 z_{S'}$

So, to evaluate this equation and also to update $S' \leftarrow S' + Q'$, we implemented Algorithm 3 with A, D, and N are the used temporary variables..
Let $X_1,\ Y_1,\ Z_3$ the projective coordinates of S' in $E'(\mathbb{F}_{p^2})$, $X_2,\ Y_2,\ Z_2$ the projective coordinates of Q' in $E'(\mathbb{F}_{p^2})$ and $X_3,\ Y_3,\ Z_3$ the projective coordinates of $S' + Q'$. We consider that the line joint S and Q evaluated on P is $l_{S,Q}(P) = t_0 + t_1\gamma + t_3\gamma^3$.

Algorithm 3:

```
Input: $x"_P = -x_P, y_P$                    Complexity                          Complexity
$X_1,\ Y_1,\ Z_3, X_2,\ Y_2,\ Z_2$
Output: $X_3,\ Y_3,\ Z_3$
$t_0,\ t_1,\ t_2$

1. $t_0 \leftarrow X_2Y_1$          $M_2$        13. $A \leftarrow A + t_1$     $A_2$
2. $t_1 \leftarrow X_1Y_2$          $M_2$        14. $A \leftarrow A - t_0$     $A_2$
3. $t_3 \leftarrow t0 - t1$         $A_2$        15. $A \leftarrow A - t_0$     $A_2$
4. $A \leftarrow X_2Z_2$            $M_2$        16. $X_3 \leftarrow DA$        $M_2$
5. $D \leftarrow X_1 - A$           $A_2$        17. $Y_3 \leftarrow t_1Y_1$    $M_2$
6. $A \leftarrow Y_2Z_2$            $M_2$        18. $t_0 \leftarrow t_0 - A$   $A_2$
7. $N \leftarrow Y_1 - A$           $A_2$        19. $t_0 \leftarrow Nt_0$      $M_2$
8. $t_0 \leftarrow D^2$             $S_2$        20. $Y_3 \leftarrow t_0 - Y_3$ $A_2$
9. $t_1 \leftarrow Dt_0$            $M_2$        21. $Z_3 \leftarrow t_1Z_1$    $M_2$
10. $t_0 \leftarrow t_0X_1$         $M_2$        22. $t_0 \leftarrow y_PD$      $2M$
11. $A \leftarrow N^2$              $S_2$        23. $t_1 \leftarrow x_P"N$     $2M$
12. $A \leftarrow AZ_1$             $M_2$
```

The global cost of this algorithm which allows as to compute the line $l_{S,Q}$ and the addition of T and P is $11M_2 + 2S_2 + 4M + 8A_2$. We need also to add the cost of the update $f \leftarrow fl_{S,Q}$ which is $15M_2 + 21A_2 + 4A'_2$ for computing the addition step in Miller loop. So, the total cost of the addition step in Miller's algorithm is $26M_2 + 2S_2 + 4m + 29A_2 + 4A'_2$.

2.3 Final Exponentiation

The final exponentiation has become the most significant parameter of the overall cost of the pairing. This step consists on the fact that a Miller loop result must be raised to the power $\frac{p^k-1}{r}$. Our paper is based on the implementation of a new variants of the final exponentiation presented by Duquesne et al. in [2]. Recall that the final exponentiation can be broken down into three components as follow.

In our case $k = 12$, so the final exponent becomes

$$\frac{p^{12}-1}{r} = \left(p^6 - 1\right)\left(p^2 + 1\right)\frac{p^4 - p^2 + 1}{r}$$

This is the natural decomposition used for the calculation of the final exponentiation. There are certainly many methods in the literature which allow us to compute this part

Table 1: Important result of computing the hard part of the final exponentiation.

Method	Cost in $\mathbb{F}_p$	Cost saving	Temp. var. in $\mathbb{F}_p$	Memory saving
Naive	$25671\,M + I$		34	
Lucas Sequence	$I + 22903\,M$		46	
Devigili	$3938\,M + 2\,I$		70	
Their variant	$\mathbf{3711\,M + 3\,I}$	$\mathbf{5.75\%}$	**58**	$\mathbf{17\%}$
Addition chain	$3366\,M + 3\,I$		130	
Their variant	$\mathbf{3363\,M + 3\,I}$	$\mathbf{-0.1\%}$	**82**	$\mathbf{37\%}$
Fuentes method	$3324\,M + 3\,I$		82	
Their variant	$\mathbf{3318\,M + 3\,I}$	$\mathbf{0,2\%}$	**70**	$\mathbf{15\%}$
New multiple	$\mathbf{3591\,M + 3\,I}$	$\mathbf{-6.4\%}$	**58**	$\mathbf{29\%}$

of the pairing, but in our implementation we are interested by reducing memory usage. For this reason, we will present our implementation results of new variants presented by Duquesne et al in [2]. We will not present their studied methods, but we will just present its final results in the table 1; which give a comparison between Duquesne et al. [2] results and results given in the literature.

After studying mathematical aspect and finding the appropriate arithmetic optimization, we will apply the resulting algorithm and the hardware techniques to present an efficient hardware design, in the following section where we will detail all components used to compute Optimal Ate Pairing.

3 Pairing Processor Design

In this section, we will present our proposal optimal ate pairing hardware design. It is based on two steps: Miller Loop and Final Exponentiation. Based on Algorithm 1, the major operations to compute optimal ate pairing are divided into two categories:

1. Dependent operations which are:
   - The point addition, point doubling, line computation $l(Q)$ (in step 1, 2, and 10), and $\phi(Q)$ (in step 9): are performed in $\mathbb{F}_{p^2}$,
   - $f^2 \times l(Q)$, $f \times l(Q)$ (in step 5 and 6 respectively) are performed in $\mathbb{F}_{p^{12}}$.
2. Independent operations which are $\mathbb{F}_p$ operations (multiplication, inversion, and square).

The data-path shown in Figure 1 uses serial/parallel approaches. According to $\mathbb{F}_{p^k}$ sub-algorithm, it executes some of the independent operations in parallel, and disabled the others in order to save resources in pairing computations. For this reason, only one $\mathbb{F}_{p^k}$ arithmetic unit ($\mathbb{F}_{p^k}$ AU) is used. It included independent operations to perform $\mathbb{F}_{p^k}$ sub-algorithms. The inputs data to the $\mathbb{F}_{p^k}$ AU come firstly from RDU, then from respective registers where results were stored. To synchronize and manage the conflicts between different blocks, a Controller Unit is needed. It send controller lines and multiplex every block in order to be used or disabled. In addition, it treat their requests and manipulate register access.

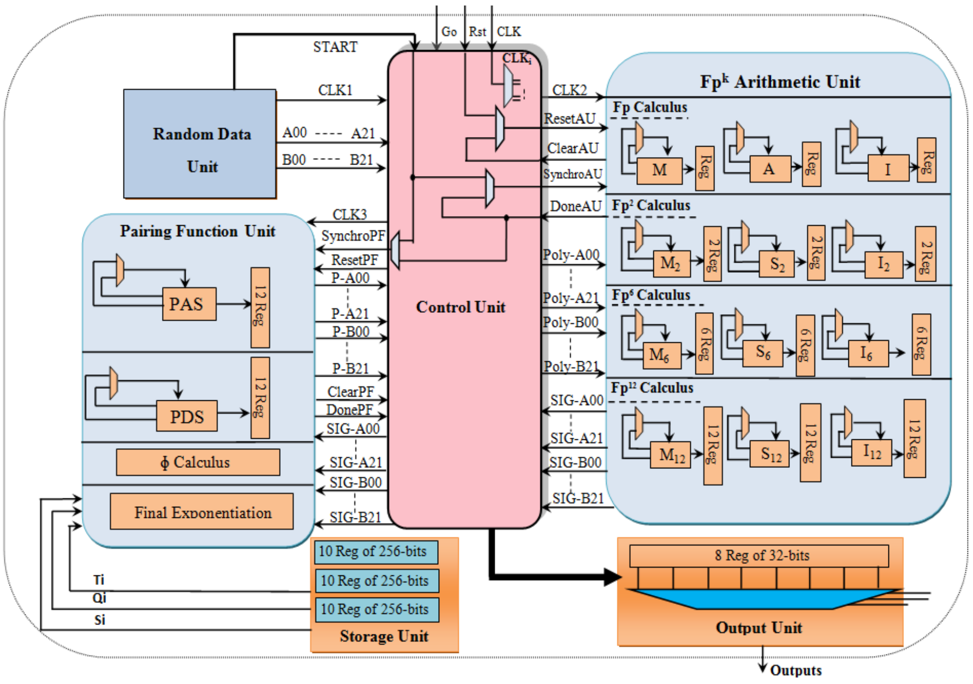

Figure 1: Pairing Computation Processor Design

Reducing the overall hardware costs of pairing computation is our first objective for designing the proposal architecture. It consists of several independent and dependent blocks which operate in serial or parallel according to the algorithm steps dependency. The whole architecture is based on five principal blocks which are:

1. **The Random Data Unit (RDU)**: generated random values are need in the pairing first step, Random Number Generator (RNG), in other high level languages, is a function of a special library. But in VHDL, RNG is achieved by designing a pseudo-random sequence generator (PRSG) of suitable length. The RDU values will be the inputs of all $\mathbb{F}_{p^k}$ Arithmetic Unit($\mathbb{F}_{p^k}$AU) and Pairing Functions Unit (PFU).

2. **The Storage Unit (SU)**: to calculate Frobenius, we have some precalculations to do, this values will be calculated once time and they will be constants during optimal ate computation. Precalculations stored in RAMs, will increase the area occupation and decrease execution time. In our design we need three different precalculations, so we will use three 256-bits-RAM. In every one we will store ten 256-bits-values.

3. **The Control Unit (CU)**: is designed to control the flow of data in the design, as well as the movement of data between registers and the execution units ($\mathbb{F}_{p^k}$AU and PFU). It is the main block of the design data path. After generating random values, the ACU is totally responsible for the system management; it coordinates

all the system blocks by sending control lines. After every $\mathbb{F}_{p^k}$ operations, it stores results in different registers to be used in the next step.

4. **The $\mathbb{F}_{p^k}$ Arithmetic Unit ($\mathbb{F}_{p^k}$AU)**: is the first block of the execution unit. It is responsible for computing all arithmetic operations in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$. All sub-algorithms are performed by $\mathbb{F}_{p^k}$AU. Firstly, it uses random values to execute first arithmetic operations, and then stores results in the different registers as it is shown in Figure 1. Stored values will be the inputs of different $\mathbb{F}_{p^k}$AU models.

5. **The Pairing Function Unit (PFU)**: it's the second block of the execution unit. It executes firstly the Miller Loop (Point Addition Step (PAS), Point Doubling Step (PDS) and ϕ calculus), then the Final Exponentiation . It uses results generated by $\mathbb{F}_{p^k}$AU models to compute base algorithms.

In the next section, we will present the hardware optimizations used in our design to find the efficient way to compute optimal ate pairing.

4 Hardware Optimization

After algorithmic optimization presented in [2], we concentrated our optimization efforts on hardware design. The problem of hardware implementation is a function of two different factors: cryptographic algorithms architectures and the efficient integration of them. The algorithms used to compute our optimal ate processor are partitioned into a sequence of hardware implementable models. Every model represents the serial behavior of the algorithm and can be executed sequentially. In this section, we will present different optimizations done to perform our architecture.

4.1 Precalculation and RAM use

Precalculation is needed to compute Frobenius calculus. Its hardware implementation occupies an important memory and increases execution time. In order to face this problem, time and memory usage can be reduced by using a storage unit capable of storing all precalculation values and constants needed in our implementation using FPGA Block RAM features. In Xilinx FPGA, we find a distributed RAM's used for implementing large and wide memory functions and it's ideal for small sized memories. Xilinx can automatically connect several distributed RAM's in parallel done by its synthesizer.

4.2 Arithmetic operation optimization

In most of the cryptosystems, there is a need of big number calculation. Operations complexities in $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$ can be expressed in term of $\mathbb{F}_p$ arithmetic. So, it necessary to explore different arithmetic functions such as multiplication, inversion, addition and square. The modular multiplication and the modular inversion are the most important operations for computing a cryptographic pairing. Until now, inversion operation doesn't have optimization and it is avoided mostly by use of projective coordinates. Moreover, squaring is considered a special case of multiplication. Whereas, researches are oriented to the modular multiplication. Many efficient multiplication algorithms had been proposed. The three most popular ones for big number multiplication are the Karatsuba-Ofman, Toom-Cook, and FFT. Every algorithm has a certain complexity, which is essentially a measure for how long it takes to run the algorithm and the difficulty of computational problems against many different computational resources such as time, area etc. Thus, to design an efficient cryptosystem, computational complexity is a primordial step to choose algorithms that are easy to implement but hard to break. Karatsuba [30], Toom [33] and Cook [34] found polynomial multiplication methods which have lower asymptotic complexity, from $O(n^2)$ to $O(n^e)$, where $1 < e \leq \log_2 3$. Many efforts have been done to find optimized implementations.

Table 2: Performance Comparison of different 256-bits Multiplier

Work	Mult.Type	Platform	Area (Slices; DSP)	Freq. (MHz)	Time (μs)
Ours	Toom-Cook-Karatsuba	Virtex6	2250; 15	145	0.89
[3]	R4MIM	Virtex6	4630	86.6	1.487
[3]	R8MIM	Virtex6	5657	71	0.93
[3]	IMM	Virtex6	3566	116	2.21
[27]	MIM on ML	Virtex6	3475	128	2
[28]	MIM on ML	Virtex6	3600	145	1.8
[26]	MR	Virtex6	4815; 12	223	-

Area occupation and running time are the most important constraints in hardware implementation. They depend on the algorithm steps. To find an efficient way to multiply two number we can apply ”Divide and conquer algorithm” which is a method for solving a problem by dividing it into different sub-problems, each one is recursively solved, and the sub-problems solutions are then combined to find the solution to the main problem. One of the good approaches is to use Toom-Cook and Karatsuba methods. Karatsuba method was used to split the input numbers into limbs of smaller size and equal width, and then expresses the larger input product in terms of calculations made on the smaller parts. Then, for the Toom-Cook multiplier, we could choose $B = 2^{31}$ or $B = 10^90$, and stored each digit as a separate 32-bit binary word. So, to maximize FPGA's resources exploitation, we used DSP features devices computing 32-bits multiplication. After performing multiplication, result should be reduced. The most used method is Montgomery reduction, specially by cryptosystems which are based on arithmetic operations modulo a large number. It is easier to be implemented in hardware, because the modulus reduction is done by shift operations avoiding the division operations (which are costly in execution time).
Table 2 compares the performance of our proposed 256-bits multiplier with different 256-bits modular multiplication implementations in the literature. Our proposed 256-bits multiplier computes one multiplication using 2250 Slices and 15 DSP in only 0.89 μs achieving a maximum frequency of 145 MHz which is less area occupation comparing to the others, and it presents the best compromise between area, frequency and execution time.

4.3 Miller Loop and Final Exponentiation optimization

Optimal ate pairing is based on Miller Loop and Final Exponentiation which need $\mathbb{F}_{p^k}$ arithmetic operations as it is shown in Figure 2. There are two ways to compute it.

First, it can be implemented using one processor performing Miller Loop then final exponentiation. Second, two separate processors can be implemented on which these two operations are pipe-lined. We should remark that two separate processors in pipeline help to reduce the computation time, but at the same time they need larger area.This paper attempts to optimize the area of the optimal ate pairing cryptoprocessor respecting a reasonable computation time. Our architecture is based on a common data-path for computing both the Miller algorithm and the Final Exponentiation one. The most famous methods to compute the hard part of the Final Exponentiation are listed and compared in Table 1. These new variant require less memory resources than the previous ones, and offer a gain of complexity with a negligible losses in execution time which makes these method very interesting for hardware implementations.

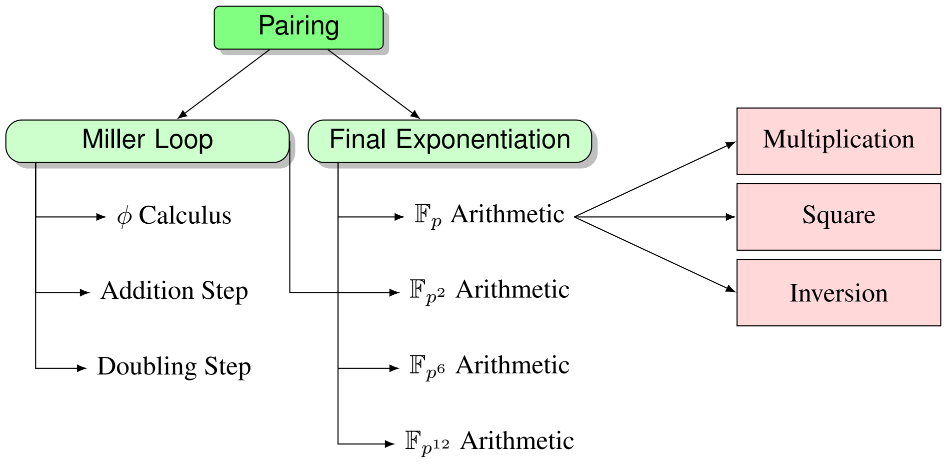
Pairing
Miller Loop
Final Exponentiation
ϕ Calculus
Addition Step
Doubling Step
$\mathbb{F}_p$ Arithmetic
$\mathbb{F}_{p^2}$ Arithmetic
$\mathbb{F}_{p^6}$ Arithmetic
$\mathbb{F}_{p^{12}}$ Arithmetic
Multiplication
Square
Inversion

Figure 2: Pairing Computation Arithmetic

As it is mentioned in Table 3, the reuse of $\mathbb{F}_{p^k}$ components can limit the number of components needed to compute every method of the hard part of the final exponentiation.

Table 3: Components Number Need for final exponentiation computation

Methods	Components Number
New development of $f^{\frac{p^4-p^2+1}{r}}$	$36A + 2M + 2S + I$
New Addition chain	$31A + 3M + 2S + I$
New development of Fuentes	$36A + 2M + 2S + I$
Variant of Fuentes	$18A + 2M + 2S + I$

Based on Table 1 and Table 3, we note that the Variant of Fuentes method is the most suitable to compute the entire Optimal Ate Pairing in term of temporary variable and memory saving.

4.4 Component Use Optimization

Our approach, to implement our architecture, is based on reuse blocks as possible as we can, because every algorithm need a defined number of arithmetic operations which can be computed in parallel or serial. Let's take the example of $\mathbb{F}_{p^6}$ multiplication in

Table 4: Clock distribution of $\mathbb{F}_{p^6}$ Multiplication Module

Steps	$\mathrm{M}\mathbb{F}_{p^2}$	Add$_1$	Add$_2$	Add$_3$	Add$_4$	Sub$_1$	Sub$_2$
1	✓	✓	✓	✓	✓	-	-
2	✓	✓	✓	✓	✓	-	-
3	✓	-	-	-	-	✓	✓
4	-	-	-	-	-	✓	✓
5	✓	✓	-	-	-	✓	-
6	✓	✓	✓	-	-	-	-
7	-	-	-	-	-	✓	✓
8	-	-	-	-	-	✓	✓
9	-	✓	✓	-	-	-	-
10	-	✓	✓	✓	-	✓	-
11	-	✓	✓	✓	✓	-	-
12	✓	-	-	-	-	✓	✓
13	-	-	-	-	-	✓	✓

Algorithm 4 (Appendix A). In every step, parallel operations can be executed in the same time, and in the next steps we can reuse the same components used before. So, here we apply the parallel approach first then the serial one and we dress Table 4. We can note that $\mathbb{F}_{p^6}$ multiplication algorithm need 6 $\mathbb{F}_{p^2}$ multiplication blocks, 14 subtraction blocks (Sub_i) and 18 addition blocks (Add_i). But, considering operation independence, we will use only 2 $\mathbb{F}_{p^2}$ multiplication blocks, 4 addition blocks and 2 subtraction blocks. The idea of activated and disabled components, can be more clear in Figure 3 (Appendix A). In every step, the component computing independent calculus was activated (with their special control lines: Go=1 and Reset=1) and the other components were disabled (Go=0 and Reset=0). We apply this approach to $\mathbb{F}_{p^{12}}$ operations.

Table 5: Block Number Optimization for computing calculus in $\mathbb{F}_{p^{12}}$

	Operation Number	Bloc Number
M_{12}	$197A + 27M$	$24A + M$
S_{12}	$148A + 18M + 27S$	$22A + M + S$
I_{12}	$201A + 59M + 29S + I$	$56A + 7M + 5S + I$

Our aim is to gain as much as possible on the number of used components, as it is mentioned in Table 5. We note that, concerning $\mathbb{F}_{p^{12}}$ operations, arithmetic block number decreased in a remarkable way. If we take the example of $\mathbb{F}_{p^{12}}$ multiplication, there was a 88$\%$ decrease in Addition blocks number , multiplication block's number was 96$\%$ lower then before. In this way, the needs of the system resources are reduced and the system performance are increased . Another point is that our architecture design is composed of several independent synchronous components which operate with their own local synchronous clocks, we have a CLK Generator in the ACU, which generate a clock to every block in the entire architecture.

5 Implementation results

We have to highlight first, that every algorithm should be demonstrated in software, such as in our case, Optimal Ate Pairing Algorithms was verified using Sage Software in [35], then being implemented in hardware. The entire architecture of the optimal ate pairing processor is coded in VHDL language. Then, the code is simulated using the Modelsim 13.1 software, synthesized using Xilinx ISE 14.7 Design Suite, and implemented on a Virtex-6 FPGA(XC6VHX250T). Our hardware design compute, first the Miller Loop then the final exponentiation. Mathematical optimization of the hard part of the Final Exponentiation has a powerful effect on the performance of the entire architecture. So, by applying: mathematical optimizations cited in [2], hardware optimizations cited in section 4, then by the use of FPGA features (Block RAMs and DSP), the architecture cost, in terms of area and memory saving, was decreased.

Table 6: Pairing Implementation results comparison

Designs	Curve	FPGA	Area Slices; DSP	Freq. (MHz)	Cycle ($\times 10^3$)
Our Design	BN_{126}	xc6vlx240t-3	5976; 30	145	80
[6]	BN_{128}	xc4vlx200-12	52K	50	821
[4]	BN_{126}	xc6vlx240t-2	7032; 32	250	143
[5]	BN_{128}	xc6vlx240t-3	4014; 42	210	245
[8]	BN_{126}	xc6vlx240t-3	5163; 144	166	62

Table 6 compares our results with the state-of-the-art implementations achieving 128-bit security. The proposed design in this paper uses 5976 Slices, 30 DSP48E1s and 3 Block RAMs, and it achieves a maximum frequency of 145 MHz. Since the number of hardware units is minimized, our design achieves the best improvement in area. By implementing our hardware design, we proved experimentally that methods computing the hard part of the final exponentiation presented in [2] necessitate less memory resources and they are in the most of time more quickly than the developments presented in the literature. Ghosh et al. [6] give the first FPGA implementation of pairings based on BN curves achieving 128-bit security, they didn't use the DSP slices on FPGA, but they used Blakley's algorithm to compute multiplication, so they have the higher area occupation with a big counts clock cycles achieving the less frequency comparing to recent implementation designs. We note here; that to speed-up pairing computations, we have to speed-up the used multiplier. Implementing pairing computation over BN curves with similar security level, our design achieves a gain of 16% in area occupation comparing to the design proposed by R. C.Cheung et all. [4]. Our clock cycle counts is drastically less, and we present a gain of 44%. J. Fan et all. [5] use, in their design,

Fp-Arithmetic operations, 74Kbit data memory and an instruction ROM implemented with FPGA Block RAMs, their design uses 4,014 Slices, 42 DSP48E1s and 5 Block RAMs (RAMB36E1s). In comparison with our results, we note that we use only 3 Block RAMs (RAMB36E1s) and 30 DSP48E1s, so our design is less costly, but design in [5] presents a gain in frequency because it achieves a maximum frequency of 210 MHz. Analyzing a result given by S. Ghosh et all. [8], we note that they use in their design a higher number of DSP blocks comparing to other designs, due to the parallelism, we used 79% less DSP in our design, but the clock cycle count of [8] is slightly reduced comparing to our results.
As a conclusion, the results of the proposal design prove that it provides significant saving of area over the existing designs.

6 Conclusion and Future Work

Efficient hardware implementation is based on optimized and demonstrated algorithms. In this paper, we implemented optimal ate pairing algorithm exploring FPGA features devices. Our architecture is based on optimized algorithms which was demonstrated in software [35]. We evaluated our design performances, we found that our implementation is less costly. These practical results prove that our optimized cryptographic architecture turned to consume as little system resources as possible, but it provided reasonable performance in the same time. For this reason, pairing implementations become more and more attractive for the hardware designers.
Nowadays a flexible encryption system, which would calculate arithmetic operations, can be implemented with hardware and software cooperation; hardware/software codesign.

References

[1] Adjej.M, Boxall.j, Duquesne.S, Greuet.A, Haloui.S, Ionica.S ElMrabet.N, Rober.D, Rondepierre.F, Sanders.O "Examples of curves, pairings and algorithms for testing on SIM cards"; $http : //simpatic.orange - labs.fr/wp - content/uploads/2013/09/ANR_INS2012_SIMPATIC_Task2_D2 - 2.pdf$.

[2] Duquesne, S., Ghammam, L.: "Memory-saving computation of the pairing final exponentiation on BN curves"; Groups Complexity Cryptology, to appear in 2016.

[3] Javeed, K., Wang, X.: "Radix-4 and Radix-8 Booth Encoded Interleaved Modular Multipliers over General Fp"; Field Programmable Logic and Applications (FPL), 24th International Conference on, 1-6 (Sep 2014).

[4] Cheung, R. C., Duquesne, S., Fan, J., Guillermin, N., Verbauwhede, I., Yao, G. X.: "FPGA implementation of pairings using residue number system and lazy reduction"; pp. 421-441, in Proc. 13th Int. Workshop CHES, Springer, Heidelberg 2011.

[5] Fan, J., Vercauteren, F., Verbauwhede, I.: "Efficient Hardware Implementation of Fp-arithmetic for Pairing-Friendly Curves"; IEEE Trans. Comput.vol.61, no. 5, pp. 676-685, (May 2012).

[6] Ghosh, S., Mukhopadhyay, D., Chowdhury, D.R.: “High Speed Flexible Pairing Cryptoprocessor on FPGA Platform”; of Lecture Notes in Computer Science, pp. 450-466, 2010.

[7] Han, J., Li, Y., Yu, Z., Zeng, X.: “A 65 nm Cryptographic Processor for High Speed Pairing Computation”; IEEE Transactions on Very Large Scale Integration (VLSI) Sstems, Vol.23, NO.4, (Avril 2015).

[8] Ghosh, S., Verbauwhede, I., Roychowdhury, D.: “Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform”; Springer-Verlag Berlin Heidelberg, 2013.

[9] Fan, J., Vercauteren, F., Verbauwhede, I.: “Faster Fp-arithmetic for cryptographic pairings on Barreto-Naehrig curves”; Int. Workshop CHES,in Proc. 11th , LNCS 5747, pp. 240-253. Lausanne, Switzerland 2009.

[10] Kammler, D.,Zhang, D., Schwabe, P., Scharwaechter, H., Langenberg M., Auras D., Ascheid G., Mathar, R.: “Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves”; Cryptographic Hardware and Embedded Systems-CHES 2009 (11th International Workshop Lausanne, Switzerland, September 6-9, 2009. Proceedings). Springer, 2009.(Lecture Notes in Computer Science; 5747).p. 254-271.

[11] Beuchat, J.L., Gonzalez-Diaz, J. E., Mitsunari, S., Okamoto, E., Rodriguez-Henriquez, F., Teruya, T.: “High-speed software implementation of the optimal ate pairing over Barreto-Naehrig curves”; Proc. 4th Int. Conf. Pairing-Based Cryptography 2010, LNCS 6487,pp. 21-39,Palo Alto, CA, USA.

[12] Vercauteren, F.: “Optimal pairings”; IEEE Trans. Inf. Theory, vol. 56, no. 1, pp. 455-461, (January 2010).

[13] Devegili, A. J., Scott, M., Dahab, R.:“Implementing cryptographic pairings over Barreto-Naehrig curves”; Pairing ’07, LNCS 4575, pp.197-207, 2007.

[14] Mitusunari, S.:“A Fast Implementation of the Optimal Ate Pairing over BN curve on Intel Haswell Processor”; Cryptology ePrint Archive, (June 2013).

[15] Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.:“Secure Dual-Core Cryptoprocessor for Pairings Over Barreto-Naehrig Curves on FPGA Platform”; IEEE Transaction on Very Large Scale Integration Systems, vol.21, no.3, (March 2013).

[16] Shyam, V., Sujatha, D.:“FPGA Implementation of an Efficient and Highly Secure Cryptoprocessor over Barreto-Naehrig Curves”; Green Computing Communication and Electrical Engineering (ICGCCEE), 2014 International Conference on, pp.1-5, 2014.

[17] Sakiyama, K., Preneel, B., Verbauwhede, I.:“A fast dual-field modular arithmetic logic unit and its hardware implementation”; Circuits and Systems, 2006 IEEE International Symposium on. IEEE. pp. 4-pp, 2006.

[18] Tenca, A. F., Koc, C. K.:“A scalable architecture for montgomery nultiplication”; Cryptographic Hardware and Embedded Systems. Springer, pp. 94-108, 1999.

[19] Al-Khaleel, O., Papachristou, C., Wolff, F., Pekmestzi K.:“Fpga based design of a large moduli multiplier for public-key cryptographic systems”; Computer Design ICCD . International Conference on. IEEE, pp. 314-319, 2006.

[20] Mondal, A., Ghosh, S., Das, A., Chowdhury D. R.:“Efficient fpga implementation of montgomery multiplier using dsp blocks”; Progress in VLSI Design and Test, pp. 370-372, Berlin, Heidelberg, Springer-Verlag, 2012.

[21] Huang, M., Gaj, K., El-Ghazawi, T.:“New hardware architectures for montgomery modular multiplication algorithm”; IEEE Transactions on computers, vol. 60, no. 7, pp. 923-936, 2011.

[22] Bunimov, V., Schimmler, M.:“Area and time efficient modular multiplication of large integers”; Application-Specific Systems, Architectures, and Processors, Proceedings. IEEE International Conference on. IEEE, 2003.

[23] Unterluggauer, T., Wenger, E.:“Efficient Pairings and ECC for Embedded Systems”; IACR and to Springer-Verlag, 2014.

[24] Xiaoxu Yao, G., Fan, J., Cheung, R. C. C., Verbauwhede, I.:“Faster Pairing Coprocessor Architecture”; Pairing 2012, Springer-Verlag Berlin Heidelberg, LNCS 7708, pp. 160-176, 2013.

[25] Xilinx:“Logicore ip block gnenrator”; p. htt://www.xilinx.com., 2010.

[26] Aranha, D. F., Karabina, K., Longa, P., Gebotys, C. H., Lopez, J.:“Faster explicit formulas for computing pairings over ordinary curves”; in Proc. 30th Annu. Int. Conf. Theory Appl. Cryptographic Eurocrypt, LNCS 6632, pp. 48-68, Tallinn, Estonia 2011.

[27] Chavez Corona, C., Moreno, E. F., Henriquez, F. R.:“Hardware design of a 256-bit prime field multiplier suitable for computing bilinear pairings”; in Reconfigurable Computing and FPGAs (ReConFig), 2011 International Conference on. IEEE, pp. 229-234, 2011.

[28] Ghosh, S., Mukhopadhyay, D., Roychowdhury, D.:“Petrel: Power and timing attack resistant elliptic curve scalar multiplier based on programmable gf(p) arithmetic unit”; Circuits and Systems I: Regular Papers, IEEE Transactions on, vol. 58, no. 8,pp. 1798-1812, (Aug 2011).

[29] Nogami, Y., Akane, M., Sakemi, Y., Katou, H., Morikawa, Y.:“Integer variable chi-based ate pairing”; In Pairing- Based Cryptography - Pairing 2008, Second International Conference, Egham, UK, September 1-3, 2008. Proceedings, pages 178-191, 2008.

[30] Cook, S. A.:“On the minimum computation time of functions”; PhD thesis, Department of Mathematics, Harvard University, 1966.

[31] Barreto, P. S. L. M., Naehrig, M.:“Pairing-friendly elliptic curves of prime order”; In Selected Areas in Cryptography, 12th International Work- shop, SAC 2005, Kingston, ON, Canada, August 11-12, 2005, Revised Selected Papers, pages 319-331, 2005.

[32] Aranha, D. F., Barreto, P. S. L. M.,Longa, P., Ricardini, J. E.:“The realm of the pairings”; In Selected Areas in Cryptography - SAC 2013 - 20th International Conference, Burnaby, BC, Canada, August 14-16, 2013, Revised Selected Papers, pages 3-25, 2013.

[33] Aranha, D. F., Karabina, K., Longa, P., Gebotys, C. H., Lòpez, J.:“Faster Explicit Formulas for Computing Pairings over Ordinary Curves”; Cryptology ePrint Archive, Report 2010/526, 2010,http://eprint.iacr.org/.

[34] Toom, A. L.:“The complexity of a scheme of functional elements realizing the multiplication of integers”; Soviet Mathematics Doklady, 3:714-716, 1963.

[35] Duquesne, S., Ghammam, L. https://cloud.sagemath.com/projects/332de229-174f-4d90-ae79-ca9d3b0fc1f7/files/Algorithms.sagews.

[36] Li, Y., Han , J., Wang, S., Fang, D., Zeng, X.:“An 800 Mhz cryptographic pairing processor in 65 nm CMOS”; in Proc. IEEE A-SSCC, Kobe, Japan, pp. 217-220, (November 2012).

Appendix A:

```
Algorithm 4:
Input: $a_{00}$, $a_{01}$, $a_{10}$, $a_{11}$, $a_{20}$, $a_{21}$, $c_{00}$, $c_{01}$, $c_{10}$, $c_{11}$, $c_{20}$, $c_{21}$, p
Output : $t_8, t_9, t_6, t_7, t_0, t_1$

Step1:
$t_2, t_3 = M_2(a_{10}, c_{10}, a_{11}, c_{11})$
$V_1, V_2, V_3, V_4 = (a_{10}+a_{20}, c_{10}+c_{20}, a_{11}+a_{21}, c_{11}+c_{21})$
Step2:
$t_6, t_7 = M_2(V_1, V_2, V_3, V_4)$
$V_1, V_2, V_3, V_4 = (a_{00}+a_{10}, c_{00}+c_{10}, a_{01}+a_{11}, c_{01}+c_{11})$
Step3:
$t_4, t_5 = M_2(a_{20}, c_{20}, a_{21}, c_{21})$
$t_6, t_7 = (t_6-t_2, t_7-t_3)$
Step4:
$t_6, t_7 = (t_6-t_4, t_7-t_5)$
Step5:
$t_8, t_9 = (t_6-t_7, t_6+t_7)$
$t_0, t_1 = M_2(a_{00}, c_{00}, a_{01}, c_{01})$
Step6:
$t_8, t_9 = (t_8+t_0, t_9+t_1)$
$t_6, t_7 = M_2(V_1, V_2, V_3, V_4)$
Step7:
$t_6, t_7 = (t_6-t_0, t_7-t_1)$
Step8:
$t_6, t_7 = (t_6-t_2, t_7-t_3)$
Step9:
$t_6, t_7 = (t_6+t_4, t_7+t_5)$
Step10:
$t_6, t_7 = (t_6-t_5, t_7+t_4)$
$V_1, V_2 = (a_{00}+a_{20}, c_{00}+c_{20})$
Step11:
$t_0, t_1 = (t_0+t_4, t_1+t_5)$
$V_3, V_4 = (a_{01}+a_{21}, c_{01}+c_{21})$
Step12:
$t_0, t_1 = (t_0-t_2, t_1-t_3)$
$t_2, t_3 = M_2(V_1, V_2, V_3, V_4)$
Step13:
$t_0, t_1 = (t_2-t_0, t_3-t_1)$
```

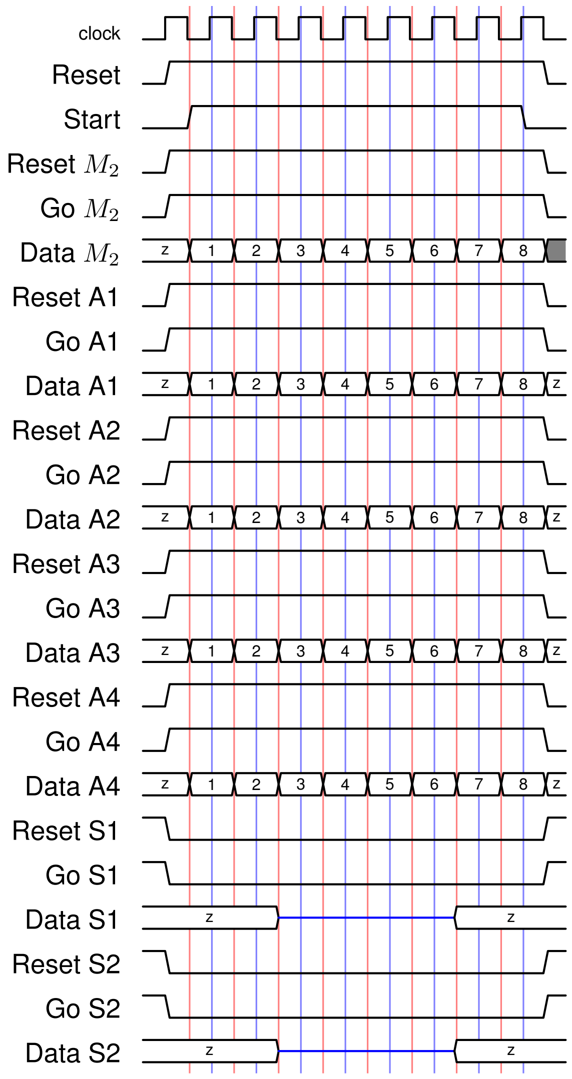

Figure 3: Chronogram of Step1 of the $\mathbb{F}_{p^6}$ Algorithm