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

Anissa Sghaier, Ghammam Loubna, Medien Zeghid
Sylvain Duquesne and Mohsen Machhout

12th November 2015

Abstract

To have an efficient asymmetric key encryption scheme, such as elliptic curves, hyperelliptic curves, pairing ... etc we have to go through arithmetic optimization then hardware optimization. Regarding 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 [1]. 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 [1], 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 9476 Slices, 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, Final Exponentiation, Miller Loop, Arithmetic optimization, Asymmetric Encryption, 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 $\mathrm{Div}(f_{u,P}) = u(P) - ([u]P) - (u-1)(P_\infty)$ where u is an

*This work was supported in part by Laboratory of electronic and microelectronic of Monastir and Henri Lebesgue center in France.

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 its hamming. After computing the Miller loop $f_1 = f_{u,P}(Q)$, we have to raising 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. [3] 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. [4] 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. [7] speed up optimal ate pairing computation having 126-bit security by exploiting IP cores available in modern FPGAs and they present a pipelined datapaths for $\mathbb{F}_p$-operations.
Until now, many hardware and software implementations was presented and 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 focus on two important mathematical results, first optimized algorithms presented by Duquesne et al. in [1] was implemented, these algorithms compute the hard part of the final exponentiation, which are New Development of $f^{\frac{p^4-p^2+1}{r}}$, New Addition Chain, Variant of Fuentes Method and New Multiple of the final exponentiation, and then we implemented a new way of computing Miller loop.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. For this reason, our hardware design is more suited for environment which allocate a huge amount of storage or which require an extensive computation power. We verified that our design is the most performing in term of area and cycle number.
In this paper, we are interested, firstly, 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 [1] 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 [1] and that is more efficient than others implementations presented in the literature and that is 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 the 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.

2 Background of pairings

2.1 BN curves presentation

Barreto and Naherig presented in [30] 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

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

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 2.1 *Nogami et al. [28] 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:

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

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:

2.2 Miller loop

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

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

For computing f we have the following algorithm1 [31]:

Points doubling and corresponding their line evaluations dominate the cost of Miller loop. Also the additions points with their corresponding line evaluations

Algorithm 1 : Optimal Ate pairing on general BLS curves

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** $i = \lfloor log_2(s) \rfloor - 2$ **downto** 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

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, projective coordinates, ...
The choice of projective coordinates has proven especially advantageous at the 128-bit security level for single pairing computation [31] and it is our case in this paper.
Now we present at first the projective coordinates. Then to perform step 5 in Miller algorithm, we present the way of the computation of $l_{Q,Q}(P)$ and the doubling step. Also to compute step 6 in algorithm 1 we have to compute $l_{S,Q}(P)$ and the addition.
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:

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

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

Computation of $l_{Q,Q}(P)$ and the doubling step:

The slope of the tangent at S is

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

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 $S_{12} + 15M_2 + 21A_2 + 4A_2'$.

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 [33]:

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

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

- $X_3 = 2x_T y_T z_T^2(y_T^2 - 9(1-i)z_T^2)$
- $Y_3 = z_T^2\left((y_T^2 + 9(1-i)z_T^2)^2 - 108(1-i)^2 z_T^4\right)$
- $Z_3 = (2y_T z_T)^3$

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 getting 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 the following algorithm:

Algorithm 2: **Input:** $x'_P = -3x_P, y_P$ $X_1,\ Y_1,\ Z_3$ **Output:**$X_3,\ Y_3,\ Z_3$ $t_0,\ t_1,\ t_2$	Computed Terms	Complexity
Temp. var used: $T_1,\ T_2,\ T_3$		
$T_1 \leftarrow X_1^2$		S_2
$T_2 \leftarrow Y_1^2$		S_2
$T_3 \leftarrow Z_1^2$		S_2
$X_3 \leftarrow X_1 + Y_1$		A_2
$X_3 \leftarrow X_3^2$		S_2
$X_3 \leftarrow X_3 - T_1$		A_2
$X_3 \leftarrow X_3 - T_2$	$2X_1Y_1$	A_2
$t_1 \leftarrow Y_1 + Z_1$		A_2
$t_1 \leftarrow t_1^2$		S_2
$t_1 \leftarrow t_1 - T2$		A_2
$t_1 \leftarrow t_1 - T3$	$2Y_1Z_1$	A_2
$t0 \leftarrow (1-i)T_3$		$\tilde{A}_2$
$t_3 \leftarrow 2t_0$		A_2
$t_0 \leftarrow t_0 + t_3$	$3(1-i)Z_1^2$	A_2
$Y_3 \leftarrow 2t_0$		A_2
$t_3 \leftarrow t_0 + Y_3$	$9(1-i)Z_1^2$	A_2
$Z_3 \leftarrow T_2 - t_3$	$Y_1^2 - 9(1-i)Z_1^2$	A_2
$X_3 \leftarrow X_3Z_3$		M_2
$t_3 \leftarrow T_2 + t_3$		A_2
$t_3 \leftarrow t_3^2$	$(Y_1^2 + 9(1-i)Z_1^2)^2$	S_2
$Y_3 \leftarrow Y_3^2$	$36(1-i)^2Z_1^4$	S_2
$t_3 \leftarrow t_3 - Y_3$	$(Y_1^2 + 9(1-i)Z_1^2)^2 - 36(1-i)^2Z_1^4$	A_2
$Y_3 \leftarrow 2Y_3$		A_2
$Y_3 \leftarrow t_3 - Y_3$		A_2
$Z_3 \leftarrow T_2t_1$	$2Y_1^3Z_1$	M_2
$Z_3 \leftarrow 2Z_3$		A_2
$Z_3 \leftarrow 2Z_3$	$8Y_1^3Z_1$	A_2
$t_3 \leftarrow T_2 - t_0$		A_2
$t_0 \leftarrow y_Pt_1$	$2Y_1Z_1y_P$	$2m$
$t_1 \leftarrow x'_PT_1$	$-3x_PX_1^2$	$2m$

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 $2XY = (X + Y)^2 - X^2 - Y^2$. So in this 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:

$$l_{S,Q}(P) = y_P - y_Q - \lambda_{S,Q}(x_P - x_Q)$$

$$= y_P - \frac{x_P(y_{S'} - y_{Q'}z_{S'})}{x_{S'} - x_{Q'}z_{S'}}\gamma + \left(\frac{x_{Q'}(y_{S'} - y_{Q'}z_{S'})}{x_{S'} - x_{Q'}z_{S'}} - y_{Q'}\right)\gamma^3$$

$$= y_P - \frac{x_P N'_1}{x_{S'} - x_{Q'}z_{S'}}\gamma + \left(\frac{x_{Q'}y_{S'} - y_{Q'}x_{S'}}{x_{S'} - x_{Q'}z_{S'}}\right)\gamma^3$$

$$= \frac{N'_2}{D'_2}$$

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 add 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 the following algorithm: 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 $l_{S,Q}(P) = t_0 + t_1\gamma + t_3\gamma^3$. So we present the following algorithm:

Algorithm 3: **Input:** $x"_P = -x_P, y_P$ $X_1,\ Y_1,\ Z_3$ $X_2,\ Y_2,\ Z_2$ **Output:**$X_3,\ Y_3,\ Z_3$ $t_0,\ t_1,\ t_2$	Computed Terms	Complexity
Temp. var used: $A,\ D,\ N$		
$t_0 \leftarrow X_2Y_1$		M_2
$t_1 \leftarrow X_1Y_2$		M_2
$t_3 \leftarrow t0 - t1$	$X_2Y_1 - X_1Y_2$	A_2
$A \leftarrow X_2Z_2$		M_2
$D \leftarrow X_1 - A$	$X_1 - X_2Z_1$	A_2
$A \leftarrow Y_2Z_2$		M_2
$N \leftarrow Y_1 - A$	$Y_1 - Y_2Z_1$	A_2
$t_0 \leftarrow D^2$		S_2
$t_1 \leftarrow Dt_0$	D^3	M_2
$t_0 \leftarrow t_0X_1$	D^2X_1	M_2
$A \leftarrow N^2$		S_2
$A \leftarrow AZ_1$		M_2
$A \leftarrow A + t_1$		A_2
$A \leftarrow A - t_0$		A_2
$A \leftarrow A - t_0$	$N^2Z_1 + D^3 - 2D^2X_1$	A_2
$X_3 \leftarrow DA$		M_2
$Y_3 \leftarrow t_1Y_1$	$D3Y_1$	M_2
$t_0 \leftarrow t_0 - A$		A_2
$t_0 \leftarrow Nt_0$	$N(D^2X_1 - A)$	M_2
$Y_3 \leftarrow t_0 - Y_3$	$N(D^2X_1 - A) - D^3Y_1$	A_2
$Z_3 \leftarrow t_1Z_1$		M_2
$t_0 \leftarrow y_PD$		$2m$
$t_1 \leftarrow x_P"N$		$2m$

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 consist on the fact that a Miller loop result must be raised to the power $\frac{p^k-1}{r}$.
Our paper is based in the implementation of a new variants of the final exponentiation presented by Duquesne et al. in [1].
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 alow us to compute this part 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 [1]
In this paper, we will not present the methods studied in [1], but we will just present its final results in the table 1, which give a comparison between Duquesne et al. [1] results and results given in the literature.
After studying mathematical aspect and finding the appropriate arithmetic op-

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	$\mathbf{3711\,M + 3\,I}$	**5.75**%	**58**	**17**%
Addition chain	$3366\,M + 3\,I$		130	
Their variant	$\mathbf{3363\,M + 3\,I}$	$\mathbf{-0.1}$%	**82**	**37**%
Fuentes method	$3324\,M + 3\,I$		82	
Their variant	$\mathbf{3318\,M + 3\,I}$	$\mathbf{0,2}$%	**70**	**15**%
New multiple	$\mathbf{3591\,M + 3\,I}$	$\mathbf{-6.4}$%	**58**	**29**%

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

timization, we will apply the resulting algorithm and the hardware techniques to present an efficient hardware design, in the following section.

3 Pairing Processor Design and implementation

Until know, Pairings is massively used in cryptographic applications. Optimal-ate pairing is an alternatives derived from the Tate pairing which is the original one. It is the most efficient one computed over elliptic curves (E) defined over a large prime field $\mathbb{F}_p$, up to date. Many algebraic curves have been used for providing efficient pairing computation technique and achieving higher security, called pairing-friendly curves, the most popular one is BN curves. So one of the best choices is to implement the optimal-ate pairing on BN curves which is the case of our work. This section presents the proposed hardware design to compute the memory saving of optimal ate pairing. This design is based on two steps: Miller Loop and Final Exponentiation.

3.1 Processor Design

In this part, we will introduce the properties of the proposal design and its implementation. A block based top-level implementation of our proposed processor is shown in figure 1. The optimal ate pairing is based on Final Exponentiation and Miller Loop. To compute Miller Loop, we need Point Addition (PA) and Point Doubling (PD). Whereas, the Final exponentiation requires operations

calculated in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$.
The proposed design necessitates five based blocks:

- **Random Data Unit (RDU)**: to compute optimal ate pairing we need random values generated in the 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 Function Unit (PFU).
- **The Data Access Unit (DAU)**: is designed to control the flow of data in the design, as well as the movement of data between registers and the Execution Unit. It contains an Access Control Unit (ACU) which coordinates all the system operations and operands. After the first step of random generation values, the ACU is totally responsible for the system management. It defines the proper constants and operation word length, it manages the ROM blocks and it controls all modules. After computing $\mathbb{F}_p^k$ operation, it stores all results in different registers to be used in the next step.
- **Storage Unit (SU)**: In our design we have some pre-calculation functions, so we need to store functions's output values to be used later. RAM model allows the storage of the different calculated values and other constants such as curve parameters. Then, we will read the appropriate value from different RAM locations.
- $\mathbb{F}_p^k$ **Arithmetic Unit ($\mathbb{F}_p^k$AU)**: is the first part of Execution Unit (EU). It represents one of the main data path component of the system architecture. This unit is responsible of computing all arithmetic operations in $\mathbb{F}_p$, $\mathbb{F}_{p^2}$, $\mathbb{F}_{p^6}$ and $\mathbb{F}_{p^{12}}$. It executes and coordinates all models and then sends the results to be stored in registers and then reused in next calculus.
- **Pairing Function Unit (PFU)**: it's the second part of the execution Unit, it contains all principal units to compute pairing. This unit includes the models that compute: the four methods of hard part of the final exponentiation mentioned in [1], the final exponentiation and Miller loop algorithms which are Addition Step and Doubling Step. The PFU uses models of $\mathbb{F}_p^k$AU to compute optimal ate pairing.

Figure 1 resume what's it mentioned earlier about the functionality of every component, we will explain data flow of our design. First of all, RDU will generate twelve values A00, A01, A10, A11, A20, A21, B00, B01, B10, B11, B20, B21, every one is of the length 256 bits. This block will be executed only once, then it send all values and a signal "START" = 1 to the ACU to begin control the rest of component. Receiving the necessary Data, ACU generate different CLK_i, as a need, and send them to the SU, $\mathbb{F}_p^k$AU and PFU. The SU should prepare all stored values (the precalculated values) to be used by $\mathbb{F}_p^k$AU and PFU. To compute all pairing function, we need all calculus done by $\mathbb{F}_p^k$AU, so that Final Exponentiation calculus or Miller Loop calculus need all $\mathbb{F}_p^k$ operations, most of the operations in PFU are performed in $\mathbb{F}_{p^{12}}$. In our design which is based on mathematical optimization [1], we tried to use lower number of addition/subtraction, multiplications and squaring, because all $\mathbb{F}_p^k$

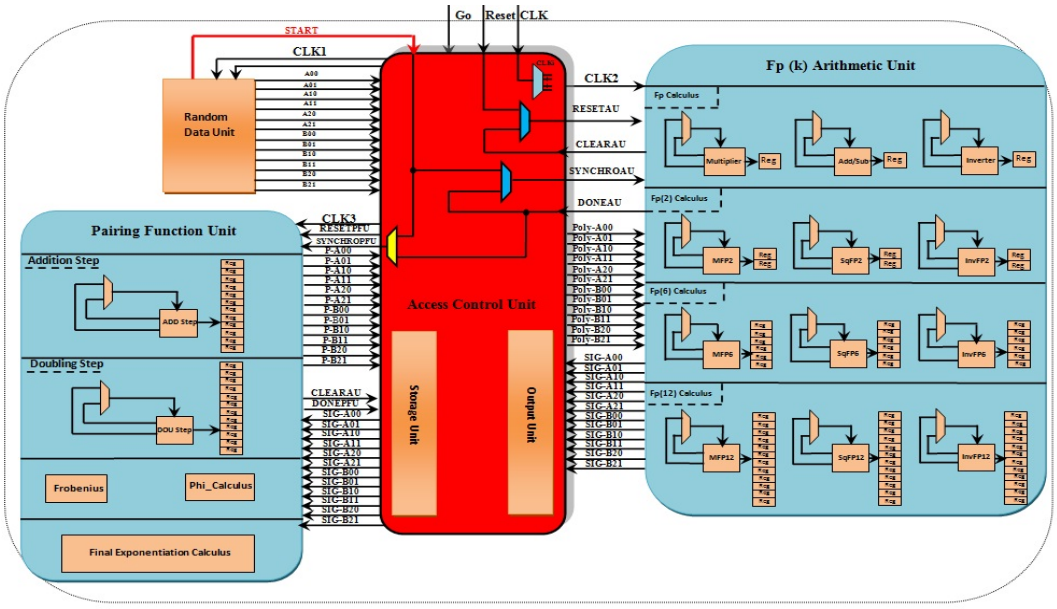

Figure 1: Hardware Processor Design

operations are based on operations computed in the base field $\mathbb{F}_p$. Here in our design, if we exploit parallelism, data-flow will be more complex and demands large memory use. The proposed design tries to make a balance between speed, area, and design complexity. We tried to do stage pipeline to provide a suitable design which compute one optimal-ate pairing at a time.
And as it's shown in figure 1, throughout our implementation we did a scheduling between the pipeline approach, the serial approach and parallelism, to maximize the utilization of individual components which certainly speeds up the pairing computation specially with constrained resources. So, in order to maximize components utilization, each of the component inputs is multiplexed between the output of ACU to be used as possible as we can. And the output of each of functional units will be stored in the register to be used later.
We should note here that the ACU have a strong role in the data-path design because to provide control signals and scheduling of operations on different functional units are very hard to do, so the ACU functions is performed by a strong state machine.
In the following section, the basic units of our architecture are described.

3.2 Processor Implementation

Our hardware design is based on hybrid architecture, our approach consist on components full time function. If calculus is independent, we use parallel approach and we use serial approach where calculus is dependant. In this case, the outputs of each operation is redirected to registers and reused for the next operation.

1. **The Data Access Unit (DAU)**
   To ensure the communication and to resolve access conflicts between RDU,

the different registers and the $\mathbb{F}_p^k$AU, DAU should be well placed and well used. The main block of DAU is Access Control Unit (ACU), his goal is to control all other modules by sending select lines to the appropriate model. The ACU select the RDU to send its outputs to be the inputs of $\mathbb{F}_p^k$AU. After doing the needed calculus, $\mathbb{F}_p^k$AU sent results to be stored in different registers. After that, every $\mathbb{F}_p^k$AU output's will be the inputs of $\mathbb{F}_p^k$AU in the next step.

2. **Storage Unit(SU)**
   To calculate Frobenius, we have some pre calculation to do, this values will be calculated once time and they will be constant during optimal ate computation. Precalculation blocks will take increase the area occupation and decrease execution time. So we tried to pick up execution time and minimize area that we lose if we do the precalculation during optimal ate computing. For this reason, we used RAM to store these values. In our design we need three different precalculation, in every one we will store 10 values written in 256 bits, so we will use three RAM-256-bits.

3. $\mathbb{F}_p^k$ **Arithmetic Unit($\mathbb{F}_p^k$AU)**
   It allows computing different $\mathbb{F}_p^k - operations$ like it shown in Figure 2, it take the input data from RDU in first step then from respective registers in the other steps. It contains all arithmetic operations blocks which are addition/subtraction, square, multiplication, inversion in $\mathbb{F}_p^k$. In theory only addition and subtraction take minimal time and area. Whereas, multiplication and inversion module take a lot of processor time and area. But in our case, we compute modular addition and subtraction, so we should take this into consideration. In addition, we need Frobenius calculus and Rapid exponentiation calculus. Figure 2, presents $\mathbb{F}_p^k$ arithmetic

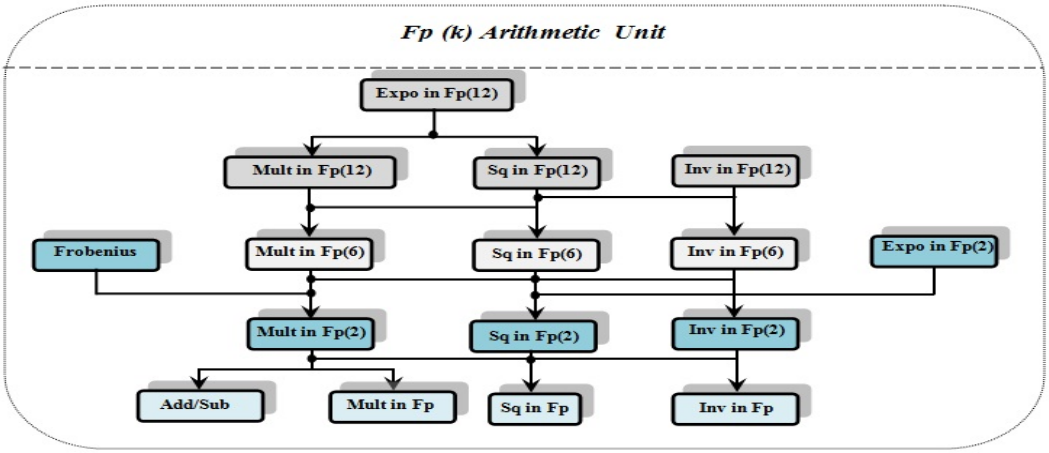

Figure 2: $\mathbb{F}_p^k$ *Arithmetic Unit*

operations needed to our implementation. To compute $\mathbb{F}_{p^{12}}$ operations, we need $\mathbb{F}_{p^6}$ operations which need $\mathbb{F}_{p^2}$ operations, and all $\mathbb{F}_p^k$ operations are based on $\mathbb{F}_p$ ones. Let's take the example of inversion computation in

$\mathbb{F}_{p^{12}}$, this operation need at the same time: multiplication in $\mathbb{F}_{p^6}$, square in $\mathbb{F}_{p^6}$ and inversion in $\mathbb{F}_{p^6}$.

4. **Pairing Function Unit**
   PFU execute all algorithms needed to compute Optimal Ate Pairing. In first time, it compute Miller Loop as it mentioned in figure 3, by calculation of Phi-Calculus, Addition Step and doubling Step. These operation need $\mathbb{F}_p$ operations. In second time, it calculate hard part of final exponentiation then the final exponentiation, as it's shown in figure 1.

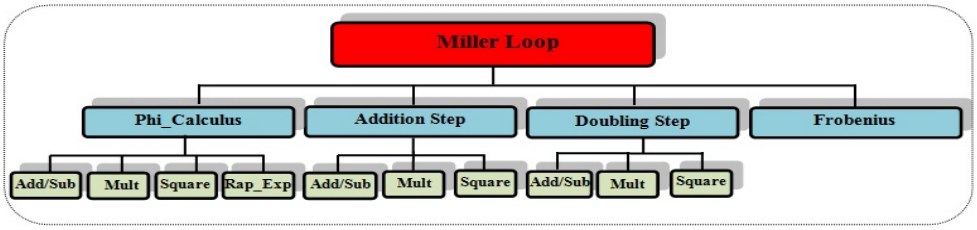

Figure 3: *Miller Loop Computation*

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 [1],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 Pre calculation and RAM use

Pre calculation is needed to compute Frobenius calculus. It need four modular additions, four modular subtraction, two modular multiplication and two modular square. All these modules occupies an important memory and increase execution time, and we should note that the covered area resource is one of the most important factors to be considered. In order to face this problem, execution time and memory usage can be reduced by using a storage unit capable of storing all pre calculation values and constants needed in our implementation using FPGA Block RAM features.
In Xilinx FPGA, we find two types of RAM: block RAM's and distributed RAM's. The first type is a dedicated two port memory containing several kilo bits of RAM. The configuration logic blocks (CLB) in most of the Xilinx FPGA's contain small single port or double port RAM which is normally distributed throughout the FPGA and it is called "distributed RAM". With this type, for

implementing larger and wider memory functions we can connect several distributed RAM's in parallel which automatically done by the Xilinx synthesizer. As we can see from the definitions, distributed RAM is ideal for small sized memories. But when comes to large memories, this may cause a extra wiring delays. But Block RAM's are fixed RAM modules so if we implement a small RAM with a block RAM it will be wastage of the rest of the space in RAM. So use block RAM for large sized memories and distributed RAM for small sized memories or FIFO's.
In other hand, we can note that the calculation is not always parallel, it's executed sequentially. This means that data acquisition can be done step by step so we don't need larger number of I/O devices for the system.

4.2 Arithmetic operation optimization

In most of the cryptosystems, there is a need of big number calculation, as it's mentioned in table 2, 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.
we should note that modular multiplication and inversion in base field are the

Operation	Notation	Cost in $\mathbb{F}_p$
Multiplication in $\mathbb{F}_p$	M	M
Squaring in $\mathbb{F}_p$	S	S
Inversion in $\mathbb{F}_p$	I	I
Multiplication in $\mathbb{F}_{p^2}$	M_2	$3M$
Squaring in $\mathbb{F}_{p^2}$	S_2	$2M$
Multiplication in $\mathbb{F}_{p^6}$	M_6	$18M$
Squaring in $\mathbb{F}_{p^6}$	S_6	$12M$
Inversion in $\mathbb{F}_{p^6}$	I_6	$37M + I$
Multiplication in $\mathbb{F}_{p^{12}}$	M_{12}	$54M$
Cyclotomic Squaring in $\mathbb{F}_{p^{12}}$	S_{12}	$18M$
Inversion in $\mathbb{F}_{p^{12}}$	I_{12}	$15M$

Table 2: Operation cost in the extension tower [1]

most important operations for computing a cryptographic pairing. Until now, inversion operation doesn't have optimization, so computing inversion is based on Fermat and Euclid algorithms, 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.
Multiplication is the process of repeating additions of the same number, it's the most costly operation after inversion, for this reason it was well studied by researches, and many different and efficient multiplication algorithms had been proposed. The three most popular big number multiplication algorithms 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 primordial step to choose algorithms that are easy to implement but hard to break. Karatsuba [29], Toom [33] and

Cook [32] 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.

Area occupation and running time are the most important computational re-

Work	Mult.Type	Platform	Field Size	Area	Freq.(MHz)	Time (μs)
Ours	Toom-Cook-Karatsuba	Virtex6	256	2250 Slices and 15 DSP	145	0.89
[2]	R4MIM	Virtex6	256	4630 Slices	86.6	1.487
[2]	R8MIM	Virtex6	256	5657 Slices	71	0.93
[2]	IMM	Virtex6	256	3566 Slices	116	2.21
[26]	MIM on ML	Virtex6	256	3475 Slices	128	2
[27]	MIM on ML	Virtex6	256	3600 Slices	145	1.8
[25]	MR	Virtex6	256	4815 Slices and 12 DSP	223	-

Table 3: Performance Comparison of different Multiplier

sources in hardware implementation especially in restrained environment, and they depend on the basic steps taken by an algorithm. Thereby, 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 combine Karatsuba method with Toom-Cook one, which is the case of our work, these two ways to efficiently multiply polynomials and long integers are well-known and well studied in the literature. 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 the multiplication was performed by applying Toom-Cook method, 32-bit multiplier, we could choose $B = 2^{31} = 2,147,483,648$ or $B = 10^9 = 1,000,000,000$, and store each digit as a separate 32-bit binary word, in this way Toom-Cook multiplier allows hardware implementation performance with increase of the execution time and fewer shifts, in this way, we should propose technique to maximize FPGA's resources exploitation available and their features devices to speed up pairing computation, so we can use the FPGA 32-bit Multiplier.

After performing multiplication, for each Fp-multiplication, we will reduce the result, here the difficulty is in the reductions modulo p, which are, essentially, division operations, and they are costly in execution time, so we need an ef-

ficient algorithm to do the reduction. The most used method is Montgomery reduction, specially by cryptosystems which are based on arithmetic operations modulo a large number. Montgomery algorithm is easier to implement in hardware,because the modulus reduction is done by shift operations avoiding the division operations.
But we should note that modular addition is required, because computing modular additions and multiplications can use the same cells for both purposes without having a full-length array of multiplexors. Therefore, these two operations have a common parts with others, so in multiplication block we need modular addition/subtraction block.
Table 3 compare the performance of our proposed multiplier with different modular multiplication implementations in the literature. Our proposed multiplier compute 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 other, and it presents the best compromise between area, frequency and execution time.

4.3 Miller Loop and Final Exponentiation optimization

Optimal ate pairing computation consists of two operations: the Miller's algorithm and the final exponentiation. There are two proposed method to compute optimal ate pairing, first it can be implemented using one processor to compute Miller Loop then final exponentiation, second two separate processors can be implemented on which these two operations are pipelined. We should remark that two separate processors in pipeline helps to reduce the computation time, but, at the same time it needs larger area. But in our implementation, we are interested to applications demanding area-constrained devices, so it is important to take care of the overall area requirement for pairing computation respecting a reasonable computation time.
This paper attempts to optimize the area of the optimal ate pairing cryptoprocessor. Our architecture is based on a common datapath for computing both the Miller algorithm and the final exponentiation. Our architecture design uses adequate parallelism and serialization in the datapath to achieve a reasonable speed cryptoprocessor with a good area-constrained devices. The proposed architecture design shown in figure 1 first computes the Miller's Algorithm then the final exponentiation.
Applying Miller's Algorithm in section 2, and the fact of reuse needed component, we have a gain in area occupation.
When speaking about the Final Exponentiation, the most famous methods

Methos	Components Number
New development of $f^{\frac{p^4-p^2+1}{r}}$	$24A + 12S + 2M + 2Sq + 1Inv$
New Addition chain	$19A + 12S + 3M + 2Sq + 1Inv$
New development of Fuentes	$24A + 12S + 2M + 2Sq + 1Inv$
Vriant of Fuentes	$12A + 6S + 2M + 2Sq + 1Inv$

Table 4: Components Number Need for final exponentiation computation

to compute its hard part was detailed and developed in [1], to obtain a new variants which are: a new development of $f^{\frac{p^4-p^2+1}{r}}$, a new addition chain, a

new development of Fuentes method and a variant of Fuentes based on a new multiple of the final exponentiation. 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 implementations in restricted environments.
As it mentioned in table 4, by the reuse of $\mathbb{F}_p^k$ components, we can limit the number of component needed to compute every method of the hard part of the final exponentiation. Based on the results cited in table 1, we note that the method of Divigili is the most suitable to restricted environment in term of temporary variable and memory saving. We implemented the four methods cited below and we proved this results, so in the final design we used Divigili method to compute the entire Optimal Ate Pairing.

4.4 Component Use Optimization

Our pairing processor architecture included a $\mathbb{F}_p^k$ Arithmetic Unit capable of computing modular additions/subtractions, modular multiplication, modular squaring and modular inversion. Moreover, squaring is considered a special case of multiplication, and inversion is avoided mostly by use of projective coordinates.
Our approach, to implement our processor, 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 algorithm 2:

Algorithm 2 resumes all steps needed to compute $\mathbb{F}_p^6$ multiplication, so we remark that we have in every step some parallel operations which can be executed in the same time, and if we pass to the next step we can reuse the same component used before. So here we apply the parallel approach firstly then the serial one.

Applying the algorithm 2, we dressed table 5 which lists the components

Steps	**Multip$\mathbb{F}_p^2$**	**Add1**	**Add2**	**Add3**	**Add4**	**Sub1**	**Sub2**
1	✓	✓	✓	✓	✓	-	-
2	✓	✓	✓	✓	✓	-	-
3	✓	-	-	-	-	✓	✓
4	-	-	-	-	-	✓	✓
5	✓	✓	-	-	-	✓	-
6	✓	✓	✓	-	-	-	-
7	-	-	-	-	-	✓	✓
8	-	-	-	-	-	✓	✓
9	-	✓	✓	-	-	-	-
10	-	✓	✓	✓	-	✓	-
11	-	✓	✓	✓	✓	-	-
12	✓	-	-	-	-	✓	✓
13	-	-	-	-	-	✓	✓

Table 5: *Clock distribution of* $\mathbb{F}_p^6$ *Multiplication Module*

activated and desactivated in every step of $\mathbb{F}_{p^6}$ multiplication algorithm. In addition, it give us, in every step, how many component we used.
The idea of activated and desactivated components, can be more clear in figure 4, in every step, component that compute independent calculus was activated with their special clock generated by a CLK Generator Block. And the other

Algorithm 2 : $\mathbb{F}_p^6$ Multiplication

Input : a00, a01, a10, a11, a20, a21, c00, c01, c10, c11, c20, c21,p
Output :t8,t9,t6,t7,t0,t1
Step1:
t2,t3=Multiplication $\mathbb{F}_p^2$(a10,c10,a11,c11)
V1, V2, V3, V4 = (a10+a20,c10+c20,a11+a21,c11+c21)
Step2:
t6,t7=Multiplication $\mathbb{F}_p^2$(V1,V2,V3,V4)
V1, V2, V3, V4 = (a00+a10,c00+c10,a01+a11,c01+c11)
Step3:
t4,t5=Multiplication $\mathbb{F}_p^2$(a20,c20,a21,c21)
t6, t7 = (t6-t2 , t7-t3)
Step4:
t6, t7 = (t6-t4 , t7-t5)
Step5:
t8,t9=(t6-t7, t6+t7)
t0,t1=Multiplication $\mathbb{F}_p^2$(a00,c00,a01,c01)
Step6:
t8,t9=(t8+t0 , t9+t1)
t6,t7=Multiplication $\mathbb{F}_p^2$(V1,V2,V3,V4)
Step7:
t6,t7=(t6-t0,t7-t1)
Step8:
t6,t7=(t6-t2,t7-t3)
Step9:
t6, t7= (t6+t4, t7+t5)
Step10:
t6, t7=(t6-t5 , t7+t4)
V1, V2 = (a00+a20,c00+c20)
Step11:
t0,t1=(t0+t4, t1+t5)
V3, V4 = (a01+a21,c01+c21) **Step12:**
t0,t1=(t0-t2, t1-t3)
t2,t3=Multiplication $\mathbb{F}_p^2$(V1,V2,V3,V4)
Step13:
t0,t1=(t2-t0, t3-t1)

components were deactivated. We can note also that $\mathbb{F}_p^6$ multiplication need 6 $\mathbb{F}_p^2$ multiplication blocks, 14 subtraction blocks and 18 addition blocks. But, considering operation independency, we will use only 2 $\mathbb{F}_p^2$ multiplication blocks, 4 addition blocks and 2 subtraction blocks.

This approach is very efficient and presents a remarkable gain in area, for this reason we will apply it in all algorithms implementations. If we take the example of $\mathbb{F}_{p^{12}}$ operations, we count the number of operations needed to compute every algorithm then analyzing parallelism and serialism approach, we found the exact number of necessary blocks to be reused in every step. Our aim was to gain as much as possible on the number of used components, as it is mentioned in table 6. We note that, concerning $\mathbb{F}_{p^{12}}$ arithmetic operations, arithmetic block

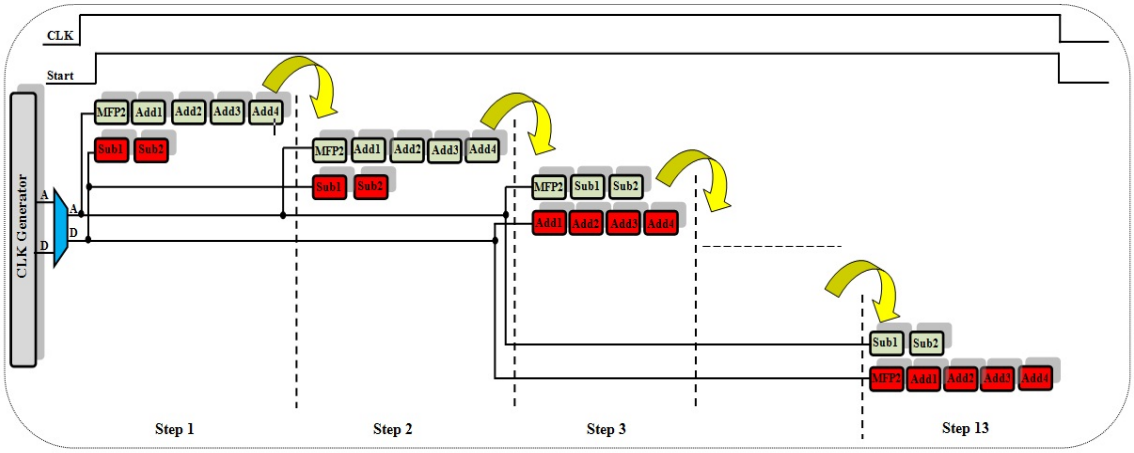

Figure 4: Timing for computing $\mathbb{F}_{p^6}$ Multiplication

	Operation Number	Bloc Number
Mult$\mathbb{F}_{p^{12}}$	$115A + 82S + 27M$	$13A + 11S + 1M$
Sq$\mathbb{F}_{p^{12}}$	$66A + 82S + 18M + 27Sq$	$10A + 12S + 1M + 1Sq$
Inv$\mathbb{F}_{p^{12}}$	$99A + 102S + 59M + 29Sq + 1Inv$	$25A + 31S + 7M + 5Sq + 1Inv$
Expo$\mathbb{F}_{p^{12}}$	$172A + 164S + 45M + 27Sq$	$23A + 23S + 1M + 1Sq$

Table 6: Block Number Optimization for computing calculus in $\mathbb{F}_{p^{12}}$

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 , subtraction Blocks's number was 86% lower then before and multiplication Blocks number decreased by 96% . 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, as you can see in figure 1, we have a CLK Generator in the ACU, which generate a clock to every block in the entire architecture. In this case, each component can adjust its operation speed by reporting its workload, even pause when not seeking.

5 Implementation results

We have to highlight firstly that every algorithm should be demonstrated in software, such is our case, Optimal Ate Pairing Algorithms was verified using Sage Software in [34], 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, compiled and synthesized using Xilinx ISE 14.7 Design Suite, then it is implemented on a Virtex-6 FPGA(XC6VHX250T)
Our hardware design compute the Miller Loop then the final exponentiation.

Mathematical optimization of the hard part of the Final Exponentiation and the doubling step in the Miller Loop has a powerful effect on the performance of the entire architecture. So, by applying: mathematical optimizations cited in section 2 and in [1], hardware optimizations cited in section 4, then by the use of FPGA features like Block RAMs on the FPGA to implement data memory and instruction ROM, the architecture cost, in terms of area and memory saving, was decreased.
Let's now remember that hardware resources on an FPGA are indicated by the number of slices that FPGA has, where a slice is comprised of look-up tables (LUTs) and flip flops. The number of LUTs and flip flops Virtex-6 FPGA contains 4 LUTs and 8 Flip Flops.
We implemented our design in two way, the first without use of DSP Slices and then using DSP Slices and Block RAMs. In the first way, on a Xilinx Virtex-6 FPGA (XC6VHX250T), we find that our hardware design uses 9476 Slices achieving a maximum frequency of 145 MHz, since the number of hardware units is minimized it also achieves the best improvement in area.
And then, by using Block RAMs and DSP Slices, the design uses 6240 Slices, 30 DSP48E1s and 3 Block RAMs. The design achieves a maximum frequency of 160 MHz. Table 7 compares the result with the state-of-the-art implementations achieving 128-bit security. All performance measurements show that our results is the best in term of area and execution time.
The mentioned results prove that our proposed architecture provide significant

Design	**Pairing**	**Security**	**Technology**	**Area**	**Freq.**	**Cycle**
Our Design	Optimal Ate	126	Xilinx FPGA Virtex 6	9476 Slices	145	80486
[5]	Optimal Ate	126	Xilinx FPGA Virtex 4	52K Slices	50	821000
[3]	Optimal Ate	126	Xilinx FPGA Virtex 6	7032 Slices and 32 DSP	250	143111
[4]	Optimal Ate	126	Xilinx FPGA Virtex 6	4014 Slices et 42 DSP	210	245430
[7]	Optimal Ate	126	Xilinx FPGA Virtex 6	5163 Slices et 144 DSP	166	62000

Table 7: Implementation results comparison

saving of area over the existing designs and it can be suitable for restricted environment because it have the tradeoff between area and time complexities.
By implementation our hardware design, we proved experimentally that methods computing the hard part of the final exponentiation presented in [1] necessitate less memory resources and they are in the most of time more quickly than the developments presented in the literature. Thinking about the more implemented method which is presented by Scott et al. and based on the addition chain requires at least 4,16 Ko of memory to be implemented on a smart card

but by their variant 2,36 Ko of a memory are largely enough to implement a pairing in a restricted environment.

6 Conclusion and Future works

6.1 Conclusion

Every algorithm should be optimized firstly in mathematical point of view, then should be demonstrated in software [34] before being implemented in hardware. In this paper, we studied hardware implementation of the optimal ate pairing algorithm optimized in [1], and demonstrated in software, then we gave an efficient and optimized architecture which explore FPGA features devices. Then we evaluated its performance and we noted that optimizations presented below certainly make the implementation less costly.

These practical results touch resource constrained systems, which can benefit greatly from our optimized cryptographic architecture that are turned to consume as little system resources as possible but it provide reasonable performance in the same time. For this reason pairing implementations become more and more attractive for the hardware designers.

As mentioned earlier, presented results show that we have the best performance in area and practical frequency in comparison with other hardware designs.

6.2 Future work

Nowadays a flexible encryption system, which would calculate arithmetic operations, can be implemented with hardware and software cooperation; hardware/software codesign.

Compared with the other hardware devices like ASICs and FPGAs, smart cards have limited computing power and minimized storage capacity, we can implement our design on this embedded system.

References

[1] Sylvain Duquesne and Loubna Ghammam, *Memory-saving computation of the pairing final exponentiation on BN curves*, IACR Cryptology ePrint Archive, Volume 2015.

[2] Khalid Javeed and Xiaojun Wang, *Radix-4 and Radix-8 Booth Encoded Interleaved Modular Multipliers over General Fp*, 2014.

[3] 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*, pp. 421-441, in Proc. 13th Int. Workshop CHES, Springer, Heidelberg.

[4] Junfeng Fan, Frederik Vercauteren, and Ingrid Verbauwhede, *Efficient Hardware Implementation of Fp-arithmetic for Pairing-Friendly Curves*, IEEE Trans. Comput.vol. 61, no. 5, pp. 676-685, May 2012.

[5] S. Ghosh, D. Mukhopadhyay, and D.R. Chowdhury, *High Speed Flexible Pairing Cryptoprocessor on FPGA Platform*, of Lecture Notes in Computer Science, pp. 450-466, 2010.

[6] Jun Han, Yang Li, Zhiyi Yu and Xiaoyang Zeng, *A 65 nm Cryptographic Processor for High Speed Pairing Computation*, IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, Vol.23, NO.4, Avril 2015.

[7] Santosh Ghosh, Ingrid Verbauwhede, and Dipanwita Roychowdhury, *Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform*, Springer-Verlag Berlin Heidelberg, 2013.

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

[9] D. Kammler et al., *An 800 Mhz cryptographic pairing processor in 65 nm CMOS*,in Proc. IEEE A-SSCC, Kobe, Japan, pp. 217-220, Novembre 2012.

[10] J.-L. Beuchat, J. E. Gonzalez-Diaz, S. Mitsunari, E. Okamoto, F. Rodriguez-Henriquez, and T. Teruya, *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.

[11] F. Vercauteren, *Optimal pairings*, IEEE Trans. Inf. Theory, vol. 56, no. 1, pp. 455-461, January 2010.

[12] A. J. Devegili, M. Scott, and R. Dahab, *Implementing cryptographic pairings over Barreto-Naehrig curves*, Pairing '07, LNCS 4575, pp.197-207, 2007.

[13] S. Mitusunari, *A Fast Implementation of the Optimal Ate Pairing over BN curve on Intel Haswell Processor*, June 2013.

[14] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, *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.

[15] Shyam V and Sujatha D, *FPGA Implementation of an Efficient and Highly Secure Cryptoprocessor over Barreto-Naehrig Curves*, 2014.

[16] K. Sakiyama, B. Preneel, and I. Verbauwhede, *A fast dual-field modular arithmetic logic unit and its hardware implementation*, Circuits and Systems, 2006. ISCAS 2006. Proceedings. 2006 IEEE International Symposium on. IEEE. pp. 4-pp.

[17] A. F. Tenca and C. K. Koc, *A scalable architecture for montgomery nultiplication*, Cryptographic Hardware and Embedded Systems. Springer, 1999, pp. 94-108.

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

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

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

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

[22] Thomas Unterluggauer and Erich Wenger, *Efficient Pairings and ECC for Embedded Systems*, IACR and to Springer-Verlag, 2014.

[23] Gavin Xiaoxu Yao and Junfeng Fan, *Faster Pairing Coprocessor Architecture*, M. Abdalla and T. Lange (Eds.): Pairing 2012, Springer-Verlag Berlin Heidelberg 2013, LNCS 7708, pp. 160-176.

[24] Xilinx, *Logicore ip block gnenrator*, p. htt://www.xilinx.com., 2010.

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

[26] C. Chavez Corona, E. F. Moreno, and F. R. Henriquez, *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, 2011, pp. 229-234.

[27] S. Ghosh, D. Mukhopadhyay, and D. Roychowdhury, *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.

[28] Yasuyuki Nogami, Masataka Akane, Yumi Sakemi, Hidehiro Katou, and Yoshitaka Morikawa, *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.

[29] Stephen A. Cook, *On the minimum computation time of functions.* PhD thesis, Department of Mathematics, Harvard University, 1966.

[30] Paulo S. L. M. Barreto and Michael Naehrig, *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.

[31] Diego F. Aranha, Paulo S. L. M. Barreto, Patrick Longa, and Jefferson E. Ricardini. 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.

[32] Andrei L. Toom, *The complexity of a scheme of functional elements realizing the multiplication of integers*. Soviet Mathematics Doklady, 3:714-716, 1963.

[33] Diego F. Aranha, Koray Karabina, Patrick Longa, Catherine H. Gebotys, Julio Lòpez, *Faster Explicit Formulas for Computing Pairings over Ordinary Curves*, Cryptology ePrint Archive, Report 2010/526, 2010,http://eprint.iacr.org/.

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