RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Coprocesador de multiplicación en $\mathbb{F}_{p^2}$ para la aceleración de emparejamientos bilineales en SoC-FPGA

Raudel Cuiman Márquez, Alejandro J. Cabrera Sarmiento, Santiago Sánchez-Solano

RESUMEN / *ABSTRACT*

El presente trabajo aborda el desarrollo de un coprocesador hardware para acelerar aquellas operaciones de multiplicación en la extensión de campo $\mathbb{F}_{p^2}$ involucradas en el cálculo de un emparejamiento bilineal. A partir de identificar el alto grado de paralelismo presente en los diferentes niveles de procesamiento aritmético de un emparejamiento, sobre todo para el caso de la multiplicación en $\mathbb{F}_{p^2}$, se propone una arquitectura hardware para el coprocesador basada en estructuras de *pipeline* tanto internas como externas que permiten acelerar el cálculo de una operación de multiplicación y habilitar, además, la ejecución de varias multiplicaciones de manera solapada. Gracias a esto ha sido posible desarrollar una solución híbrida hardware/software sobre un SoC-FPGA para el cálculo de emparejamientos bilineales que logra mejorar hasta en un 22.5% los resultados de soluciones equivalentes en el estado del arte.

Palabras claves: emparejamientos bilineales, aritmética de torre de campos, FPGA, aceleración hardware.

This paper focuses on the implementation of a hardware coprocessor intended to speed up multiplications over the $\mathbb{F}_{p^2}$ finite field extension in the context of bilinear pairings. Being aware of the high degree of parallelism present at different levels of pairings computation, especially in the case of $\mathbb{F}_{p^2}$ multiplications, we propose a hardware architecture based on internal and external pipeline structures allowing both, accelerate a single multiplication and perform several multiplications in parallel. This enables the development of a hybrid hardware/software solution on a SoC-FPGA for computing bilinear pairings that improves the performance of equivalent state-of-the-art implementations up to a 22.5%.

Keywords: bilinear pairings, tower field arithmetic, FPGA, hardware acceleration.

Multiplication coprocessor in $\mathbb{F}_{p^2}$ for bilinear pairings acceleration on SoC-FPGA

1.- INTRODUCCIÓN

Los emparejamientos bilineales sobre curvas elípticas se han erigido como una potente herramienta que permite construir diversos protocolos de seguridad para brindar vías de solución a varios problemas existentes en la criptografía. Entre los ejemplos más notables se deben mencionar el protocolo tripartito de acuerdo de claves con una sola ronda de mensajes propuesto en [1] y el protocolo de acuerdo de claves basado en identidad sin interacción entre las partes presentado en [2]. Igualmente se han dado a conocer esquemas de cifrado basado en identidad [3-5], esquemas de firma digital de pequeña longitud [6], así como esquemas de cifrado basado en atributos [7] y esquemas de cifrado homomórfico [8]. Sin embargo, el proceso de cálculo de un emparejamiento bilineal constituye una tarea compleja con un costo en cuanto a tiempo de ejecución que supera al de las operaciones involucradas en criptosistemas convencionales como RSA[1] y ECC[2]. Por ejemplo, para una

[1] RSA: Criptosistema de clave pública nombrado a partir de las iniciales de los apellidos de sus autores (Ronald L. Rivest, Adi Shamir y Leonard M. Adleman). La seguridad de RSA se basa en el problema de la factorización de números enteros.
[2] ECC: *Elliptic Curve Cryptography*. Criptosistema de clave pública basado en el problema del logaritmo discreto sobre curvas elípticas

Recibido: 28/10/2022 Aceptado: 27/12/2022

curva elíptica definida sobre un campo finito de 256 bits, se ha reportado que calcular una multiplicación escalar (operación principal en ECC) resulta hasta 30 veces más rápido que efectuar el cálculo de un emparejamiento bilineal [9,10]. Esto ha motivado el curso de numerosas investigaciones enfocadas en desarrollar formulaciones algorítmicas y evaluar estrategias de implementación que permitan obtener soluciones eficientes para el cálculo de emparejamientos bilineales. Especial atención han tenido las implementaciones para sistemas empotrados donde generalmente los recursos de procesamiento son limitados en comparación con los de una computadora de propósito general.

En la literatura se han propuesto varias implementaciones sobre sistemas empotrados de soluciones enfocadas en el cálculo de emparejamientos bilineales o de sus principales operaciones aritméticas. Algunas de estas soluciones han sido desarrolladas en software [11-15], y otras basadas en diseños hardware [16-21]. En el despliegue de las soluciones software ha predominado el empleo de procesadores ARM debido a su popularidad en el campo de los sistemas empotrados. En este caso, los mejores resultados de rendimiento en cuanto a tiempo de ejecución se han obtenido a partir de la optimización de operaciones en lenguaje ensamblador [11], el empleo de instrucciones SIMD [12-14] y de estrategias de procesamiento multinúcleo [15].

Las soluciones basadas en hardware exhiben dos tipos de implementaciones: rígidas [17,18,21] y flexibles [16,19,22,23]. Las primeras persiguen obtener buenos resultados de tiempo al optimizar las implementaciones para un tipo de emparejamiento y conjunto de parámetros específicos. En cambio, las soluciones flexibles, ofrecen cierto nivel de aceleración, pero manteniendo facilidades para implementar diferentes tipos de emparejamientos, así como modificar los parámetros asociados a la curva elíptica y a las operaciones aritméticas subyacentes. Esto constituye una ventaja importante cuando se requiere actualizar una primitiva criptográfica para satisfacer nuevos requisitos de seguridad.

Una alternativa para la obtención de implementaciones basadas en hardware con cierto grado de flexibilidad consiste en emplear estrategias de diseño híbrido hardware/software. De esta forma, se implementan en hardware aquellas operaciones básicas que sean exigentes en cuanto a tiempo, mientras que las operaciones de alto nivel y el control del flujo de procesamiento se llevan a cabo en un procesador de propósito general. Este enfoque ha sido explorado en [20] mediante la interconexión de un procesador ARM Cortex-M0+ encargado de ejecutar la parte software y un coprocesador implementado en un ASIC para la aritmética de campo finito que es acelerada mediante hardware. La arquitectura de este coprocesador se basa en el empleo de estructuras de *pipeline* internas que permiten simultanear las operaciones básicas de multiplicación y acumulación a medida que se procesan las palabras (de 32 bits) de los operandos. Con esto, los autores de [20] alcanzan un nivel de aceleración de la aritmética de campo que demuestra la pertinencia del enfoque de diseño híbrido al obtener un rendimiento seis veces superior en el cálculo de un emparejamiento respecto a una solución puramente software desarrollada sobre el propio procesador ARM Cortex-M0+.

Teniendo en cuenta lo anterior, en este trabajo se propone explotar en mayor medida la paralelización de operaciones mediante hardware extendiendo el alcance de las estructuras de *pipeline*. En este caso se trata no sólo de simultanear las operaciones de palabra internas a una operación de campo finito, sino que, a partir de identificar el alto grado de paralelismo presente en el cálculo de un emparejamiento, se propone además estructurar el diseño de un *pipeline* externo que permita solapar la ejecución de varias operaciones de campo. Igualmente se pretende explorar el empleo de un ancho de palabra de 64 bits, lo cual se traduce en una reducción del número de operaciones básicas que se requieren efectuar a nivel de palabra para completar una operación de campo finito. Estas ideas han dado origen a la implementación del coprocesador que se describe a lo largo del presente trabajo, cuyo propósito radica en desarrollar una solución híbrida hardware/software para el cálculo de emparejamientos bilineales que logre mejorar el rendimiento de soluciones equivalentes reportadas en el estado del arte.

El contenido expuesto en el resto del documento ha sido estructurado de la siguiente forma: La Sección 2 presenta una breve introducción al emparejamiento Ate óptimo sobre curvas de Barreto-Naehrig. En la Sección 3 y la Sección 4 se discuten los aspectos relacionados con la aritmética de curva y la aritmética de campo requeridas en el cálculo de un emparejamiento, enfatizando las oportunidades de paralelismo existentes en cada caso. La Sección 5 aborda el proceso de implementación del coprocesador propuesto en este trabajo, mientras que en la Sección 6 se analiza el impacto del mismo en el rendimiento de un emparejamiento. Finalmente en la Sección 7 se brindan las consideraciones finales.

2.- EMPAREJAMIENTO ATE-ÓPTIMO

La mayoría de las implementaciones prácticas enfocadas en el cálculo eficiente de emparejamientos bilineales están basadas en el emparejamiento Ate-óptimo [24]. En particular, la variante definida sobre curvas de Barreto-Naehrig [25] constituye una de las más extendidas en la literatura por ser de las alternativas más eficientes para satisfacer originalmente un nivel de seguridad de 128 bits [26].

Las curvas de Barreto-Naehrig conforman una de las familias de curvas elípticas conocidas como *curvas amigables* [27] para el cálculo de emparejamientos bilineales. En este caso se trata de curvas descritas mediante la expresión $E/\mathbb{F}_p: y^2 = x^3 + b$

que poseen orden primo r y grado de incrustación $k = 12$, donde la característica p del campo finito subyacente y el orden de la curva r son parametrizados mediante los siguientes polinomios:

$$\begin{aligned} p &= p(z) = 36z^4 + 36z^3 + 24z^2 + 6z + 1 \\ r &= r(z) = 36z^4 + 36z^3 + 18z^2 + 6z + 1 \end{aligned} \tag{1}$$

El parámetro $z \in \mathbb{Z}$ debe seleccionarse de manera que $p = p(z)$ y $r = r(z)$ sean números primos con el tamaño adecuado para satisfacer el nivel de seguridad deseado. Al mismo tiempo, es determinante seleccionar un valor de z que garantice que el primo p resultante conduzca a la construcción de la extensión $\mathbb{F}_{p^{12}}$ con propiedades favorables para realizar operaciones aritméticas de forma eficiente [28]. En este trabajo se emplea el valor $z = -4080000000000001_{16}$ propuesto en [29] como la mejor opción para proporcionar una fortaleza de 128 bits de seguridad equivalente a partir de generar valores de p y r de 254 bits.

Por otra parte, una de las propiedades más útiles de las curvas de Barreto-Naehrig radica en la existencia de una curva E' isomórfica con E que permite trasladar gran parte de las operaciones involucradas en el cálculo del emparejamiento desde $\mathbb{F}_{p^{12}}$ hacia la extensión de menor grado $\mathbb{F}_{p^2}$. La ecuación de E' es $E'/\mathbb{F}_{p^2}: y'^2 = x'^3 + b'$, con $b' = b\xi^{-1}$ o $b' = b\xi^{-5}$ donde ξ es un no residuo cuadrático y no residuo cúbico en $\mathbb{F}_{p^2}$. Sea $\alpha = \sqrt[6]{\xi} \in \mathbb{F}_{p^{12}}$, el mapa $\psi: E'(\mathbb{F}_{p^2}) \mapsto E(\mathbb{F}_{p^{12}})$ que transforma puntos de $E'(\mathbb{F}_{p^2})$ en puntos de $E(\mathbb{F}_{p^{12}})$ es $(x', y') \mapsto (\alpha^2 x', \alpha^3 y')$ [30]. De esta forma, sean E y E' una curva de Barreto-Naehrig y su curva isomórfica sobre $\mathbb{F}_{p^2}$ respectivamente y téngase $P \in E(\mathbb{F}_p)$, $Q' \in E'(\mathbb{F}_{p^2})$ junto al operador de Frobenius $\pi_p: (x, y) \mapsto (x^p, y^p)$, el emparejamiento Ate-óptimo en curvas de Barreto-Naehrig (AteOpt-BN) [24] se puede expresar como:

$$\hat{a}_{\text{opt}}(Q', P) = \left(f_{6z+2,\psi(Q')}(P) \cdot \ell_{[6z+2]\psi(Q'),\pi_p(\psi(Q'))}(P) \cdot \ell_{[6z+2]\psi(Q')+\pi_p(\psi(Q')),-\pi_p^2(\psi(Q'))}(P) \right)^{(p^{12}-1)/r} \tag{2}$$

Para materializar el cálculo de la ecuación (2) se emplea el Algoritmo 1, el cual se basa en las contribuciones realizadas por Miller en [31,32]. El bucle comprendido entre las líneas 2 y 9 se encarga de calcular el valor $f_{6z+2,\psi(Q')}(P)$. En tanto, la secuencia que se extiende desde el paso 10 hasta el paso 14 corresponde al ajuste que se aplica sobre el valor $f_{6z+2,\psi(Q')}(P)$ para garantizar que el resultado final satisfaga las propiedades no degenerativa y de bilinealidad de un emparejamiento. No es difícil verificar que al término del bucle de Miller, junto con $f_{6z+2,\psi(Q')}(P)$, se obtiene el punto $R' = [6z + 2]Q'$. Esto permite comprobar que las líneas involucradas en las multiplicaciones del tipo $f \cdot \ell_{\psi(R'),\psi(Q_1')}(P)$ de los pasos 11 y 14 en efecto corresponden a las líneas $\ell_{[6z+2]\psi(Q'),\pi_p(\psi(Q'))}(P)$ y $\ell_{[6z+2]\psi(Q')+\pi_p(\psi(Q')),-\pi_p^2(\psi(Q'))}(P)$ respectivamente.

Algoritmo 1
Algoritmo de Miller para el emparejamiento AteOpt-BN

ENTRADAS: $P \in E(\mathbb{F}_p)$, $Q' \in E'(\mathbb{F}_{p^2})$, $l = 6z + 2 = (l_{n-1}, \cdots, l_1, l_0)_2$, con $l_{n-1} = 1$.

SALIDAS: $\hat{a}_{\text{opt}}(Q', P)$

1. $R' = Q'$; $f = 1$;	**10.** $Q_1' = \pi_p(Q')$;
2. **for** $i = n - 2$ **downto** 0 **do** `// bucle de Miller`	**11.** $f \cdot \ell_{\psi(R'),\psi(Q_1')}(P)$;
3. $f = f^2 \cdot \ell_{\psi(R'),\psi(R')}(P)$;	**12.** $R' = R' + Q_1'$;
4. $R' = [2]R'$;	**13.** $Q_1' = -\pi_p(Q_1')$;
5. **if** $l_i == 1$ **then**	**14.** $f \cdot \ell_{\psi(R'),\psi(Q_1')}(P)$;
6. $f = f \cdot \ell_{\psi(R'),\psi(Q')}(P)$;	**15.** $\hat{a}_{\text{opt}} = f^{(p^{12}-1)/r}$
7. $R' = R' + Q'$;	**16.** **return** $\hat{a}_{\text{opt}}$;
8. **end**	
9. **end**	

Vale destacar que la variable f donde se va acumulando el resultado, así como las líneas secantes y tangentes que se evalúan durante la ejecución del algoritmo, son elementos de $\mathbb{F}_{p^{12}}$. No obstante, en el caso particular de las líneas, los elementos de $\mathbb{F}_{p^{12}}$ resultantes poseen la mitad de sus coeficientes con valor cero. Esto permite efectuar las operaciones del tipo $f \cdot \ell_{\cdot,\cdot}(P)$ mediante un caso especial de multiplicación denominada multiplicación dispersa que resulta más eficiente que una multiplicación genérica en $\mathbb{F}_{p^{12}}$.

2.1.- EXPONENCIACIÓN FINAL

El paso 15 del Algoritmo 1 se conoce como etapa de exponenciación final. Para acelerar el cálculo de esta etapa se ha adoptado el método propuesto en [33], el cual se basa en descomponer en factores el exponente $(p^{12}-1)/r$ en una parte denominada *exponente fácil* y otra conocida como *exponente difícil*:

$$(p^{12}-1)/r = \underbrace{(p^6-1)\cdot(p^2+1)}_{\text{exponente fácil}} \cdot \underbrace{[(p^4-p^2+1)/r]}_{\text{exponente difícil}} \tag{3}$$

El cálculo de $f^{(p^6-1)\cdot(p^2+1)}$ se divide a su vez en $g = f^{(p^6-1)} = f^{p^6}\cdot f^{-1}$ y $h = g^{(p^2+1)} = g^{p^2}\cdot g$. Dado que $f \in \mathbb{F}_{p^{12}}$ se representa como un binomio con coeficientes en $\mathbb{F}_{p^6}$, el término f^{p^6} se obtiene como el conjugado $\bar{f}$ de f. Las restantes operaciones involucradas en el cálculo de la parte fácil de la exponenciación final incluyen una inversa y dos multiplicaciones en $\mathbb{F}_{p^{12}}$, así como una exponenciación a la potencia p^2, la cual se realiza a un costo relativamente bajo mediante el operador de Frobenius. Es importante destacar que el elemento $g \in \mathbb{F}_{p^{12}}$ resultante de elevar f al exponente (p^6-1) es un elemento *unitario* [34]. Los elementos de este tipo forman un subgrupo de $\mathbb{F}^*_{p^{12}}$, por lo que cualquier operación subsiguiente que se realice sobre g resulta en otro elemento unitario. Esto resulta favorable para la implementación de la parte difícil de la etapa de exponenciación final ya que, en primer lugar, permite sustituir en lo adelante las operaciones de inversa por conjugaciones. O sea, para todo elemento unitario $g \in \mathbb{F}_{p^{12}}$ se cumple que $g^{-1} = \bar{g}$. Por otra parte, determinar el cuadrado de elementos unitarios resulta más eficiente que una operación de cuadrado genérica en $\mathbb{F}_{p^{12}}$.

Para calcular la parte difícil $h^{(p^4-p^2+1)/r}$ se representa el exponente en base p como $\chi_3 p^3 + \chi_2 p^2 + \chi_1 p + \chi_0$. Los coeficientes χ_i se expresan en función del parámetro z de Barreto-Naehrig de la siguiente forma:

$$\begin{aligned} \chi_0(z) &= -36z^3 - 30z^2 - 18z - 2 \quad \chi_2(z) = 6z^2 + 1 \\ \chi_1(z) &= -36z^3 - 18z^2 - 12z + 1 \quad \chi_3(z) = 1 \end{aligned} \tag{4}$$

A partir de la ecuación (4) se tiene que $h^{(\chi_3 p^3+\chi_2 p^2+\chi_1 p+\chi_0)}$ se convierte en:

$$\left[\underbrace{h^p \cdot h^{p^2} \cdot h^{p^3}}_{y_0}\right] \cdot \left[\underbrace{1/h}_{y_1}\right]^2 \cdot \left[\underbrace{\left(h^{z^2}\right)^{p^2}}_{y_2}\right]^6 \cdot \left[\underbrace{1/(h^z)^p}_{y_3}\right]^{12} \cdot \left[\underbrace{1/\left(h^z \cdot \left(h^{z^2}\right)\right)^p}_{y_4}\right]^{18} \cdot \left[\underbrace{1/h^{z^2}}_{y_5}\right]^{30} \cdot \left[\underbrace{1/\left(h^{z^3} \cdot \left(h^{z^3}\right)\right)^p}_{y_6}\right]^{36} \tag{5}$$

En la ecuación (5) primero se calculan los términos h^z, $h^{z^2} = (h^z)^z$ y $h^{z^3} = \left(h^{z^2}\right)^z$. Seguidamente, las operaciones de exponenciación a las potencias p, p^2 y p^3 se calculan mediante el operador de Frobenius. Las fracciones involucradas en los términos y_1 y del y_3 al y_6 representan inversas de campo que, como se ha mencionado, son sustituidas por conjugadas gracias a las propiedades de los elementos unitarios. Finalmente, la expresión $y_0 \cdot y_1^2 \cdot y_2^6 \cdot y_3^{12} \cdot y_4^{18} \cdot y_5^{30} \cdot y_6^{36}$ se evalúa de manera eficiente mediante una cadena de suma vectorial de longitud mínima [35].

3.- ARITMÉTICA DE CURVAS

Como se aprecia en el Algoritmo 1 las operaciones internas al bucle de Miller involucran la evaluación de líneas y el cálculo de puntos sobre la curva isomórfica E'. Emplear el sistema de coordenadas adecuado para representar los puntos de curva es crucial para obtener buenos resultados de rendimiento. En general, la representación de puntos en coordenadas proyectivas conduce a implementaciones de la aritmética de curva con un menor costo computacional en comparación al que se obtiene mediante el empleo de la representación tradicional en coordenadas afines [36].

Un punto $T \in E(\mathbb{F}_{p^i})$ en coordenadas proyectivas se representa como $T = (X_T : Y_T : Z_T)$ con $X_T, Y_T, Z_T \in \mathbb{F}_{p^i}$. El mapa $(X_T : Y_T : Z_T) \mapsto (X_T Z_T^{-c}, Y_T Z_T^{-d})$ permite trasladar un punto en coordenadas proyectivas a su representación en coordenadas afines (x_T, y_T). Diferentes valores para los parámetros c y d determinan sistemas de coordenadas proyectivas diferentes. Por ejemplo, los valores $c = d = 1$ definen el sistema de proyectivas estándares [37], mientras que el empleo de $c = 2$ y $d = 3$ conduce a las proyectivas Jacobianas [38]. Estas últimas son precisamente las que han sido seleccionadas en el presente trabajo debido a que en el contexto del emparejamiento AteOpt-BN conducen a formulaciones para la aritmética de puntos con un costo ligeramente menor que las que se obtienen en base a las coordenadas proyectivas estándares.

3.1.- SUMA DE PUNTOS Y LÍNEA SECANTE

Sea E una curva de Barreto-Naehrig y $E'/\mathbb{F}_{p^2}$ su curva isomórfica junto con el mapa ψ definido en la Sección 2. Téngase $P \in E(\mathbb{F}_p)$ y $Q' \in E'(\mathbb{F}_{p^2})$. Asumiendo que $T' = (X_{T'} : Y_{T'} : Z_{T'}) \in E'(\mathbb{F}_{p^2})$ en coordenadas Jacobianas representa cualquier valor intermedio del punto R' en el Algoritmo 1, la ecuación (10) permite calcular las coordenadas $(X_{R'} : Y_{R'} : Z_{R'})$ del punto $R' = T' + Q'$, mientas que la ecuación (7) corresponde al cálculo y evaluación de la línea secante $\ell_{\psi(T'),\psi(Q')}(P)$.

$$\begin{aligned} X_{R'} &= \left(Y_{T'} - y_{Q'} Z_{T'}^3\right)^2 - \left(X_{T'} + x_{Q'} Z_{T'}^2\right)\left(X_{T'} - x_{Q'} Z_{T'}^2\right)^2 \\ Z_{R'} &= Z_{T'}\left(X_{T'} - x_{Q'} Z_{T'}^2\right) \\ Y_{R'} &= \left(Y_{T'} - y_{Q'} Z_{T'}^3\right)\left(y_{Q'} Z_{R'}^2 - X_{R'}\right) - y_{Q'} Z_{R'}^3 \end{aligned} \tag{6}$$

$$\ell_{\psi(T'),\psi(Q')}(P) = \left(y_P Z_{T'}^3 - \alpha^3 Y_{T'}\right) Z_{T'}\left(X_{T'} - x_{Q'} Z_{T'}^2\right) + \left(Y_{T'} - y_{Q'} Z_{T'}^3\right)\left(\alpha^3 x_{Q'} - \alpha x_P\right) \tag{7}$$

Se puede apreciar que las ecuaciones (6) y (7) comparten varios términos. Esto permite seguir las recomendaciones de [39] para combinar en un solo procedimiento (Algoritmo 2) los procesos de cálculo de líneas secantes y suma de puntos. Todas las operaciones se realizan en $\mathbb{F}_{p^2}$ excepto las cuatro multiplicaciones de los pasos del 8 al 11 que corresponden a elementos de $\mathbb{F}_p$.

Algoritmo 2
Cálculo combinado de suma de puntos y líneas secantes

ENTRADAS: $T' = (X_{T'} : Y_{T'} : Z_{T'}) \in E'(\mathbb{F}_{p^2})$, $P = (x_P, x_P) \in E(\mathbb{F}_p)$ y $Q' = (x_{Q'}, x_{Q'}) \in E'(\mathbb{F}_{p^2})$.

SALIDAS: $\ell_{\psi(T'),\psi(Q')}(P) \in \mathbb{F}_{p^{12}}$, $R' = T' + Q'$

1. $T_0 = Z_{T'}^2$;
2. $T_1 = Z_{T'} \cdot T_0$;
3. $T_2 = x_{Q'} \cdot T_0$;
4. $T_0 = X_{T'} - T_2$;
5. $T_3 = Z_{T'} \cdot T_0$;
6. $T_1 = y_{Q'} \cdot T_1$;
7. $T_1 = Y_{T'} - T_1$;
8. $\ell_{[0][0][0]} = y_P \cdot T_{3_{[0]}}$;
9. $\ell_{[0][0][1]} = y_P \cdot T_{3_{[1]}}$;
10. $\ell_{[1][0][0]} = x_P \cdot T_{1_{[0]}}$;
11. $\ell_{[1][0][1]} = x_P \cdot T_{1_{[1]}}$;
12. $T_4 = y_{Q'} \cdot T_3$;
13. $\ell_{[1][1]} = x_{Q'} \cdot T_1$;
14. $\ell_{[1][0]} = -\ell_{[1][0]}$
15. $\ell_{[1][1]} = \ell_{[1][1]} - T_4$;
16. $T_2 = X_{T'} + T_2$;
17. $Z_{R'} = T_3$;
18. $T_3 = T_3^2$;
19. $T_0 = T_0^2$;
20. $X_{R'} = T_1^2$;
21. $T_3 = Z_{R'} \cdot T_3$;
22. $T_2 = T_0 \cdot T_2$;
23. $T_5 = x_{Q'} \cdot T_3$;
24. $X_{R'} = X_{R'} - T_2$;
25. $T_5 = T_5 - X_{R'}$;
26. $Y_{R'} = T_1 \cdot T_5$;
27. $T_3 = y_{Q'} \cdot T_3$;
28. $Y_{R'} = Y_{R'} - T_3$;
29. **return** ℓ y $R' = (X_{R'} : Y_{R'} : Z_{R'})$;

3.2.- DOBLADO DE PUNTOS Y LÍNEA TANGENTE

Partiendo de las mismas premisas del apartado 3.1 es posible definir las ecuaciones (8) y (9) para calcular las coordenadas $(X_{T\prime}: Y_{T\prime}: Z_{T\prime})$ del punto $R' = [2]T'$ y evaluar la línea tangente $\ell_{\psi(T'),\psi(T')}(P)$ respectivamente.

$$\begin{aligned} X_{R\prime} &= (3X_{T\prime}^2)^2 - 8X_{T\prime}Y_{T\prime}^2 \\ Y_{R\prime} &= 3X_{T\prime}^2(4X_{T\prime}Y_{T\prime}^2 - X_{R\prime}) - 8Y_{T\prime}^4 \\ Z_{R\prime} &= 2Y_{T\prime}Z_{T\prime} \end{aligned} \tag{8}$$

$$\ell_{\psi(T'),\psi(T')}(P) = 2Y_{T\prime}(y_P Z_{T\prime}^3 - \alpha^3 Y_{T\prime}) + 3X_{T\prime}^2(\alpha^3 X_{T\prime} - \alpha x_P Z_{T\prime}^2) \tag{9}$$

En este caso la existencia de términos comunes en ambas ecuaciones también permite formular un único procedimiento para el cálculo combinado de doblado de puntos y evaluación de tangentes sobre la curva elíptica (Algoritmo 3). De igual forma todas las operaciones se realizan en $\mathbb{F}_{p^2}$ exceptuando las cuatro multiplicaciones en $\mathbb{F}_p$ de los pasos del 11 al 14.

Algoritmo 3
Cálculo combinado de doblado de puntos y líneas tangentes

ENTRADAS: $T' = (X_{T\prime}: Y_{T\prime}: Z_{T\prime}) \in E'(\mathbb{F}_{p^2})$ y $P = (x_P, x_P) \in E(\mathbb{F}_p)$.

SALIDAS: $\ell_{\psi(T'),\psi(T')}(P) \in \mathbb{F}_{p^{12}}$, $R' = [2]T'$

1. $T_0 = X_{T\prime}^2$;
2. $T_1 = Z_{T\prime}^2$;
3. $T_2 = Y_{T\prime}^2$;
4. $T_0 = 3T_0$;
5. $\ell_{[1][1]} = X_{T\prime} \cdot T_0$;
6. $T_3 = Z_{T\prime} \cdot T_1$;
7. $T_4 = T_0 \cdot T_1$;
8. $T_5 = 2Y_{T\prime}$;
9. $T_2 = 2T_2$;
10. $T_3 = T_3 \cdot T_5$;
11. $\ell_{[0][0][0]} = y_P \cdot T_{3_{[0]}}$;
12. $\ell_{[0][0][1]} = y_P \cdot T_{3_{[1]}}$;
13. $\ell_{[1][0][0]} = x_P \cdot T_{4_{[0]}}$;
14. $\ell_{[1][0][1]} = x_P \cdot T_{4_{[1]}}$;
15. $\ell_{[1][0]} = -\ell_{[1][0]}$
16. $\ell_{[1][1]} = \ell_{[1][1]} - T_2$;
17. $T_1 = 2X_{T\prime}$;
18. $X_{R\prime} = T_0^2$;
19. $T_4 = T_2^2$;
20. $T_1 = T_1 \cdot T_2$;
21. $Z_{R\prime} = Z_{T\prime} \cdot T_5$;
22. $T_4 = 2T_4$;
23. $T_3 = 2T_1$;
24. $X_{R\prime} = X_{R\prime} - T_3$;
25. $T_1 = T_1 - X_{R\prime}$;
26. $Y_{R\prime} = T_0 \cdot T_1$;
27. $Y_{R\prime} = Y_{R\prime} - T_4$;
28. **return** ℓ y $R' = (X_{R\prime}: Y_{R\prime}: Z_{R\prime})$;

4.- TORRE DE CAMPOS

Las sentencias del tipo f^2 y $f \cdot \ell_{\cdot,\cdot}(P)$ presentes en el algoritmo de Miller corresponden a operaciones de cuadrado y multiplicación de elementos en $\mathbb{F}_{p^{12}}$. Es por ello que la forma en que se construya esta extensión de campo es crucial para obtener buenos resultados de rendimiento en el proceso de cálculo de un emparejamiento. En general, los autores de [40] han observado que si el grado k de una extensión $\mathbb{F}_{p^k}$ puede expresarse como $k = 2^i 3^j$, entonces es posible representar dicha extensión como una torre de extensiones cuadráticas y cúbicas. En particular, siguiendo las recomendaciones de [28,41], para la implementación del emparejamiento AteOpt-BN en este trabajo se emplea la torre $\mathbb{F}_p \overset{2}{\rightarrow} \mathbb{F}_{p^2} \overset{3}{\rightarrow} \mathbb{F}_{p^6} \overset{2}{\rightarrow} \mathbb{F}_{p^{12}}$ dada por:

$$\begin{aligned} &\mathbb{F}_{p^2} = \mathbb{F}_p[u]/(u^2 - \beta)\text{, donde } \beta = -1 \\ &\mathbb{F}_{p^6} = \mathbb{F}_{p^2}[v]/(v^3 - \xi)\text{, donde } \xi = u + 1 \\ &\mathbb{F}_{p^{12}} = \mathbb{F}_{p^6}[w]/(w^2 - \gamma)\text{, donde } \gamma = v \end{aligned} \tag{10}$$

Los elementos $a \in \mathbb{F}_{p^2}$, $b \in \mathbb{F}_{p^6}$ y $c \in \mathbb{F}_{p^{12}}$ se representan mediante binomios o trinomios según corresponda. De esta forma se tiene que $a = a_{[0]} + a_{[1]}u$, $b = b_{[0]} + b_{[1]}v + b_{[2]}v^2$ y $c = c_{[0]} + c_{[1]}w$, donde $a_{[0]}, a_{[1]} \in \mathbb{F}_p$, $b_{[0]}, b_{[1]}, b_{[2]} \in \mathbb{F}_{p^2}$ y $c_{[0]}, c_{[1]} \in \mathbb{F}_{p^6}$. Así, para referenciar coeficientes específicos dentro de estas representaciones polinómicas se emplea la misma

notación utilizada en [15]. Por ejemplo, en el elemento $c \in \mathbb{F}_{p^{12}}$ el coeficiente $c_{[0]} \in \mathbb{F}_{p^6}$ se representaría como $c_{[0]} = c_{[0][0]} + c_{[0][1]}v + c_{[0][2]}v^2$, con $c_{[0][0]}, c_{[0][1]}, c_{[0][2]} \in \mathbb{F}_{p^2}$, al tiempo que el coeficiente $c_{[0][0]}$ se representaría mediante el binomio $c_{[0][0]} = c_{[0][0][0]} + c_{[0][0][1]}u$, donde $c_{[0][0][0]}, c_{[0][0][1]} \in \mathbb{F}_p$.

Uno de los grandes beneficios de utilizar torres de campos radica en que las extensiones de un mismo grado comparten similitud estructural. Esto permite emplear formulaciones algorítmicas similares para implementar aquellas operaciones aritméticas equivalentes en cada una de ellas. Por ejemplo, en la torre de la ecuación (10) el método que se utilice para efectuar multiplicaciones en $\mathbb{F}_{p^2}$ puede adaptarse para multiplicar elementos en $\mathbb{F}_{p^{12}}$ ya que ambas son extensiones cuadráticas. Por otra parte, la recursividad que introducen las torres de campo conduce a que cualquier mejora en el campo base o alguna extensión intermedia se refleje directamente en una mejora de rendimiento de las operaciones en las extensiones siguientes. En este sentido, como se ha visto en las secciones 2 y 3, la mayoría de las operaciones aritméticas involucradas en el cálculo del emparejamiento Ate-óptimo en curvas de Barreto-Naehrig tienen lugar en $\mathbb{F}_{p^2}$ y $\mathbb{F}_{p^{12}}$. Además, debido a la estructura de la torre de campos $\mathbb{F}_p \xrightarrow{2} \mathbb{F}_{p^2} \xrightarrow{3} \mathbb{F}_{p^6} \xrightarrow{2} \mathbb{F}_{p^{12}}$, la aritmética en $\mathbb{F}_{p^{12}}$ se compone de operaciones aritméticas en $\mathbb{F}_{p^6}$, que a su vez están conformadas por operaciones en $\mathbb{F}_{p^2}$. Por ello, para acelerar el cálculo del emparejamiento AteOpt-BN resulta determinante partir de una implementación eficiente de la aritmética en $\mathbb{F}_{p^2}$, lo cual es válido también para la implementación de cualquier emparejamiento sobre curvas elípticas con grado de incrustación par [25,42-44].

4.1.- ARITMÉTICA EN $\mathbb{F}_{p^2}$

Las operaciones en $\mathbb{F}_{p^2}$ que se requieren implementar en la construcción de la torre de campos de la ecuación (10) son la suma y la resta, la multiplicación, el cuadrado y la inversa multiplicativa. Debido a que los elementos de $\mathbb{F}_{p^2}$ se representan como binomios del tipo $a = a_{[0]} + a_{[1]}u$ con coeficientes $a_{[0]}, a_{[1]} \in \mathbb{F}_p$, las operaciones en esta extensión involucran la realización de cálculos subyacentes en el campo base $\mathbb{F}_p$, resultando igualmente necesario implementar operaciones de suma, resta, multiplicación e inversa multiplicativa. En este caso se trata de aritmética modular con números enteros (módulo p), donde, para mayor eficiencia, se ha optado por la representación numérica y el método de reducción de Montgomery [45], así como el algoritmo extendido de Euclides [46] para el cálculo de inversas.

De la aritmética en $\mathbb{F}_{p^2}$ resaltan la multiplicación, el cuadrado y la inversa multiplicativa como las más exigentes en cuanto a costo computacional, siendo la multiplicación la que aparece con mayor frecuencia durante el cálculo de un emparejamiento. Por ejemplo, representando los costos de estas operaciones (multiplicación, cuadrado e inversa multiplicativa en $\mathbb{F}_{p^2}$) como $\widetilde{\mathbf{M}}$, $\tilde{\mathbf{C}}$ e $\tilde{\mathbf{I}}$ respectivamente, a partir de analizar el algoritmo de Miller (Algoritmo 1) es posible estimar las cantidades requeridas de cada una de estas operaciones para calcular un emparejamiento. Como se ha visto, el bucle de Miller para el emparejamiento AteOpt-BN se basa en el cálculo y evaluación de líneas tangentes y secantes de conjunto con el doblado y la suma de puntos sobre la curva isomórfica $E'/\mathbb{F}_{p^2}$, además de operaciones de cuadrado y multiplicación (multiplicación dispersa) en $\mathbb{F}_{p^{12}}$. Del Algoritmo 2 y el Algoritmo 3 es fácil apreciar que cálculo combinado de líneas secantes y suma de puntos requiere de $11\widetilde{\mathbf{M}}$ y $4\tilde{\mathbf{C}}$, mientras que en el cálculo de tangentes junto al doblado de puntos se utilizan $7\widetilde{\mathbf{M}}$ y $5\tilde{\mathbf{C}}$. Asimismo, para el cuadrado y la multiplicación dispersa en $\mathbb{F}_{p^{12}}$ se tienen métricas de $12\widetilde{\mathbf{M}}$ y $13\widetilde{\mathbf{M}}$ respectivamente (ver Algoritmo 5 y Algoritmo 6). Debido a que el valor $z = -4080000000000001_{16}$ seleccionado en la Sección 2 conlleva a que la longitud del bucle de Miller ($l = 6z + 2$) sea un número de 65 bits con un peso de Hamming de 5, en total se realizan 64 evaluaciones de tangentes y doblado de puntos, cuatro evaluaciones de líneas secantes y suma de puntos, así como 64 cuadrados y 68 multiplicaciones dispersas en $\mathbb{F}_{p^{12}}$. Esto conduce a un costo de $2144\widetilde{\mathbf{M}}$ y $336\tilde{\mathbf{C}}$ para el bucle de Miller. Realizando un análisis similar con la etapa de exponenciación final se puede obtener un costo estimado para la misma de $1780\widetilde{\mathbf{M}}$, $772\tilde{\mathbf{C}}$ y $1\tilde{\mathbf{I}}$, lo cual suma un total de $3924\widetilde{\mathbf{M}}$, $1108\tilde{\mathbf{C}}$ y $1\tilde{\mathbf{I}}$ para el emparejamiento AteOpt-BN. Esto implica que, de las operaciones en $\mathbb{F}_{p^2}$ con mayor demanda computacional, la multiplicación ocupa aproximadamente el 78% de la carga de procesamiento. Es por ello que la formulación algorítmica adecuada y la implementación eficiente de esta operación resultan cruciales para incrementar el rendimiento del proceso de cálculo de un emparejamiento.

Teniendo en cuenta lo anterior, para implementar la multiplicación en $\mathbb{F}_{p^2}$ se ha optado por la técnica de multiplicación de binomios de Karatsuba [47] en combinación con la reducción módulo $u^2 + 1$. De ahí se derivan las expresiones que se muestran en la ecuación (11) para calcular los coeficientes $c_{[0]}, c_{[1]}$ del elemento $c = a \cdot b$ [48]. En estas expresiones ya se tiene implícita la etapa de reducción considerando la equivalencia $u^2 = -1$.

$$c_{[0]} = (a_{[0]} \cdot b_{[0]}) - (a_{[1]} \cdot b_{[1]})$$
$$c_{[1]} = [(a_{[0]} + a_{[1]}) \cdot (b_{[0]} + b_{[1]})] - (a_{[0]} \cdot b_{[0]}) - (a_{[1]} \cdot b_{[1]}) \quad (11)$$

En la formulación de la secuencia algorítmica que materializa la ecuación (11) se ha tenido en cuenta, además, la posibilidad de aplicar la estrategia de reducción demorada [49] debido a que el módulo p que se obtiene al evaluar la ecuación (1) para $z = -4080000000000001_{16}$ resulta en un número de exactamente 254 bits. Esta longitud queda dos bits por debajo de 256 que es el múltiplo más cercano del ancho de palabra de 64 bits seleccionado para las implementaciones realizadas en el contexto de esta investigación. Tal diferencia abre margen para que se puedan acumular varias operaciones de multiplicación y suma de enteros (elementos de $\mathbb{F}_p$) antes de tener que aplicar una reducción modular. Esto permite disminuir la cantidad de reducciones modulares e incrementar así el rendimiento general de una multiplicación en $\mathbb{F}_{p^2}$. El Algoritmo 4 muestra los pasos detallados del procedimiento resultante. Cabe señalar que todas las operaciones de suma, resta y multiplicación que forman parte del algoritmo corresponden a operaciones convencionales de la aritmética con números enteros, a la vez que es posible notar que las operaciones de reducción modular (sólo una por cada coeficiente calculado) han sido demoradas y se efectúan solamente al final del algoritmo.

Algoritmo 4
Multiplicación de Karatsuba en $\mathbb{F}_{p^2}$ utilizando reducción demorada

ENTRADAS: $a = a_{[0]} + a_{[1]}u$ y $b = b_{[0]} + b_{[1]}u$.

SALIDAS: $c = c_{[0]} + c_{[1]}u$

1. $S_0 = a_{[0]} + a_{[1]}$;
2. $S_1 = b_{[0]} + b_{[1]}$;
3. $T_0 = a_{[0]} \cdot b_{[0]}$;
4. $T_1 = a_{[1]} \cdot b_{[1]}$;
5. $T_2 = S_0 \cdot S_1$;
6. $T_2 = T_2 - T_0$;
7. $T_2 = T_2 - T_1$;
8. $T_0 = T_0 - T_1$;
9. $c_{[0]} = T_0 \bmod p$;
10. $c_{[1]} = T_2 \bmod p$;
11. **return** $c = c_{[0]} + c_{[1]}u$;

4.2.- ARITMÉTICA EN $\mathbb{F}_{p^{12}}$

Como se ha mencionado anteriormente, las principales operaciones aritméticas en $\mathbb{F}_{p^{12}}$ involucradas en el bucle de Miller son el cuadrado y la multiplicación dispersa. Dados los elementos $a = a_{[0]} + a_{[1]}w$ y $b = b_{[0]} + b_{[1]}w$, donde $a, b \in \mathbb{F}_{p^{12}}$ con $a_{[0]}, a_{[1]}, b_{[0]}, b_{[1]} \in \mathbb{F}_{p^6}$ se formulan los siguientes algoritmos para implementar dichas operaciones:

Algoritmo 5
Cuadrado en $\mathbb{F}_{p^{12}}$

ENTRADAS: $a = a_{[0]} + a_{[1]}w$

SALIDAS: $c = c_{[0]} + c_{[1]}w = a^2$

1. $T_0 = a_{[1]} \cdot a_{[0]}$;
2. $T_1 = a_{[1]} \cdot \gamma$;
3. $T_2 = T_0 \cdot \gamma$;
4. $S_0 = a_{[1]} + a_{[0]}$;
5. $T_1 = T_1 + a_{[0]}$;
6. $S_0 = S_0 \cdot T_1$;
7. $c_{[1]} = 2T_0$;
8. $S_0 = S_0 - T_0$;
9. $c_{[0]} = S_0 - T_2$;
10. **return** $c = c_{[0]} + c_{[1]}w$;

La definición de la torre de campos de la Sección 4 implica que la aritmética en $\mathbb{F}_{p^{12}}$ se nutre de operaciones en $\mathbb{F}_{p^6}$ que a su vez involucran operaciones aritméticas en $\mathbb{F}_{p^2}$. Teniendo esto en cuenta, no es complicado expresar el costo del Algoritmo 5 en función de operaciones en $\mathbb{F}_{p^2}$. Para la aritmética en $\mathbb{F}_{p^6}$ se emplearon las expresiones para extensiones cúbicas definidas en [48] donde se puede constatar que una multiplicación en $\mathbb{F}_{p^6}$ involucra seis multiplicaciones en $\mathbb{F}_{p^2}$. Esto significa que

para un cuadrado en $\mathbb{F}_{p^{12}}$ se requieren realizar doce multiplicaciones en $\mathbb{F}_{p^2}$ (pasos 1 y 6 del Algoritmo 5). Los pasos se refieren a multiplicaciones por la constante γ que solamente involucran sumas, restas y desplazamientos con elementos de $\mathbb{F}_{p^2}$.

En el caso del Algoritmo 6 se puede observar que las operaciones se realizan sobre coeficientes en $\mathbb{F}_{p^2}$ (ver notación definida en la Sección 4), con lo cual es simple notar que este algoritmo involucra un saldo de trece multiplicaciones en $\mathbb{F}_{p^2}$.

Algoritmo 6
Multiplicación dispersa en $\mathbb{F}_{p^{12}}$

ENTRADAS: $a = a_{[0]} + a_{[1]}w$ y $b = b_{[0]} + b_{[1]}w$

SALIDAS: $c = c_{[0]} + c_{[1]}w = a \cdot b$

1. $S_0 = a_{[1][2]} + a_{[0][2]}$;
2. $S_1 = a_{[1][1]} + a_{[0][1]}$;
3. $S_2 = a_{[1][0]} + a_{[0][0]}$;
4. $S_3 = b_{[1][0]} + b_{[0][0]}$;
5. $S_4 = S_0 + S_1$;
6. $S_5 = S_3 + b_{[1][1]}$;
7. $S_6 = a_{[1][2]} + a_{[1][1]}$;
8. $S_7 = b_{[1][1]} + b_{[1][0]}$;
9. $T_0 = S_0 \cdot b_{[1][1]}$;
10. $T_1 = S_1 \cdot S_3$;
11. $T_2 = S_2 \cdot b_{[1][1]}$;
12. $T_3 = S_2 \cdot S_3$;
13. $S_4 = S_4 \cdot S_5$;
14. $T_4 = a_{[1][2]} \cdot b_{[1][1]}$;
15. $T_5 = a_{[1][1]} \cdot b_{[1][0]}$;
16. $T_6 = a_{[1][0]} \cdot b_{[1][1]}$
17. $T_7 = a_{[1][0]} \cdot b_{[1][0]}$;
18. $S_6 = S_6 \cdot S_7$;
19. $T_8 = a_{[0][2]} \cdot b_{[0][0]}$;
20. $T_9 = a_{[0][1]} \cdot b_{[0][0]}$;
21. $T_{10} = a_{[0][0]} \cdot b_{[0][0]}$;
22. $S_0 = S_4 - T_0$;
23. $S_8 = S_6 - T_4$;
24. $S_0 = S_0 - T_1$;
25. $S_8 = S_8 - T_5$;
26. $T_0 = T_0 \cdot \xi$;
27. $T_4 = T_4 \cdot \xi$;
28. $S_1 = T_1 + T_2$;
29. $S_2 = T_0 + T_3$;
30. $S_9 = T_4 + T_7$;
31. $S_{10} = T_5 + T_6$;
32. $c_{[1][0]} = S_2 - S_9$;
33. $c_{[1][1]} = S_1 - S_{10}$;
34. $c_{[1][2]} = S_0 - S_8$;
35. $c_{[1][0]} = c_{[1][0]} - T_{10}$;
36. $c_{[1][1]} = c_{[1][1]} - T_9$;
37. $c_{[1][2]} = c_{[1][2]} - T_8$;
38. $S_8 = S_8 \cdot \xi$;
39. $c_{[0][0]} = S_8 + T_{10}$;
40. $c_{[0][1]} = S_9 + T_9$;
41. $c_{[0][2]} = S_{10} + T_8$;
42. **return** $c = c_{[0][0]} + c_{[0][1]}v + c_{[0][2]}v^2 + (c_{[1][0]} + c_{[1][1]}v + c_{[1][2]}v^2)w$

5.- COPROCESADOR DE MULTIPLICACIÓN EN $\mathbb{F}_{p^2}$

Tanto en el Algoritmo 2 como en el Algoritmo 3 las operaciones de multiplicación en $\mathbb{F}_{p^2}$ se han organizado cuidadosamente de manera que exista la mayor independencia de datos posible. Lo mismo sucede con el Algoritmo 6. Como regla general, aquellas multiplicaciones en $\mathbb{F}_{p^2}$ sin dependencia de datos han sido agrupadas y dispuestas de forma consecutiva en la formulación de cada uno de esos algoritmos. Por ejemplo, las dos multiplicaciones en $\mathbb{F}_{p^2}$ de los pasos 2 y 3 del Algoritmo 2 son independientes entre sí, así como las dos de los pasos 5 y 6, las tres de los pasos del 21 al 23 y el par de los pasos 26 y 27. Algo similar sucede en el Algoritmo 3, pero es en el Algoritmo 6 donde este paralelismo resulta más notable ya que las trece multiplicaciones en $\mathbb{F}_{p^2}$ que aparecen en ese procedimiento son completamente independientes. Así, se genera la posibilidad, en cada caso, de agrupar las operaciones de multiplicación en $\mathbb{F}_{p^2}$ para ser ejecutadas de forma simultánea en tanto se disponga de la capacidad de procesamiento para ello. Esto ha sido aprovechado por los autores de [15] en una implementación software basada en procesamiento doble núcleo para acelerar el cálculo del emparejamiento AteOpt-BN agrupando y ejecutando en paralelo las operaciones de multiplicación en $\mathbb{F}_{p^2}$ de dos en dos. De esta forma lograron obtener una mejora de rendimiento de un 20% respecto a una variante de solución desplegada en un solo núcleo. No obstante, en el análisis anterior se constata que los diferentes niveles de procesamiento aritmético de un emparejamiento admiten capacidades superiores de paralelismo.

Precisamente, esta sección se propone explotar los recursos disponibles en las plataformas de hardware reconfigurable con el fin de implementar un coprocesador que acelere la operación de multiplicación en $\mathbb{F}_{p^2}$ en el contexto de los emparejamientos bilineales. Con ello se persigue desarrollar una solución híbrida hardware⁄software donde las operaciones de alto nivel involucradas en el cálculo del emparejamiento se implementen en software en un microprocesador, el cual estaría en interacción con el coprocesador para efectuar las multiplicaciones en $\mathbb{F}_{p^2}$ subyacentes y aprovechar así la posibilidad de simultanear la ejecución de varias de estas multiplicaciones.

Para el despliegue de esta solución híbrida hardware/software fue empleada la placa de desarrollo ZedBoard [50] como plataforma de pruebas. Esta placa está basada en el dispositivo SoC-FPGA Zynq-7000 XC7Z020 de Xilinx, el cual integra en un mismo encapsulado un sistema de procesamiento basado en un microprocesador ARM Cortex-A9 de doble núcleo junto a un FPGA de la serie 7 de Xilinx [51].

5.1.- PREMISAS DE DISEÑO

En esencia, el diseño del coprocesador de multiplicación en $\mathbb{F}_{p^2}$ consiste en una traducción a hardware del Algoritmo 4, donde se han tenido en cuenta tres premisas fundamentales:

1. Aprovechar el paralelismo existente en las operaciones internas de una multiplicación en $\mathbb{F}_{p^2}$.
2. Proporcionar el mayor grado de paralelismo posible al ejecutar simultáneamente varias operaciones de multiplicación en $\mathbb{F}_{p^2}$ pero manteniendo un consumo moderado de recursos hardware.
3. Gestionar la comunicación entre el coprocesador y el microprocesador de forma eficiente.

La primera premisa se refiere a que las operaciones que conforman una multiplicación en $\mathbb{F}_{p^2}$ también exhiben cierto grado de paralelismo. Es fácil notar que las sumas y multiplicaciones de números enteros indicadas en los primeros cuatro pasos del Algoritmo 4 no presentan dependencia de datos. Lo mismo sucede con las reducciones modulares de los pasos 9 y 10. Esto evidentemente permite paralelizar el cálculo de dichas operaciones en la implementación del coprocesador.

La segunda premisa expresa el objetivo principal del coprocesador, el cual, como se ha mencionado, consiste en ejecutar en paralelo varias operaciones de multiplicación en $\mathbb{F}_{p^2}$, lo cual se pudiera lograr replicando la arquitectura concebida para una multiplicación tantas veces como sea necesario. No obstante, este enfoque puede conducir a un consumo elevado de recursos hardware, lo cual, en algunos casos, puede resultar en un diseño irrealizable o con un balance poco factible entre costo y beneficio. Una alternativa que consigue minimizar el consumo de recursos consiste en diseñar una arquitectura que solape varias operaciones de multiplicación en $\mathbb{F}_{p^2}$ sobre un único bloque de procesamiento hardware. Para ello, se concibe una estructura de *pipeline* que, siguiendo el planteamiento de la primera premisa, organiza el flujo de ejecución de una multiplicación en $\mathbb{F}_{p^2}$ en varias etapas de procesamiento que actúan en paralelo. Cuando una etapa termina de procesar los datos de una operación le entrega los resultados parciales a la etapa siguiente y queda lista para procesar nuevos datos que lleguen a su entrada, los cuales corresponderán a una nueva operación. Esta cadena de procesamiento induce un efecto de paralelismo que logra reducir el tiempo necesario para obtener el resultado de cada nueva multiplicación en $\mathbb{F}_{p^2}$.

La última premisa refleja el hecho de que el rendimiento de una solución híbrida hardware/software no puede desligarse del mecanismo empleado para el intercambio de datos. La gestión eficiente de la comunicación entre el microprocesador y el coprocesador de multiplicación en $\mathbb{F}_{p^2}$ debe abordarse desde diferentes aristas. Por un lado, es importante seleccionar la interfaz de comunicación que mejor se ajuste a los requisitos de la solución en cuanto a la estructura y el volumen de los datos a transferir y, por el otro, es preciso enlazar este flujo de datos con la cadena de procesamiento de forma que no se afecte el proceso de ejecución de una multiplicación en $\mathbb{F}_{p^2}$. En este sentido, téngase en cuenta que un elemento de $\mathbb{F}_{p^2}$ está compuesto por dos elementos de $\mathbb{F}_p$, los cuales, en el marco de esta investigación, corresponden a números enteros de longitud $n = 254$ representados como secuencias de $m = \lceil n/\omega \rceil$ palabras de ω bits. En este caso para el microprocesador ARM de la plataforma Zynq-7000 se tiene que $\omega = 32$. Por tanto, un elemento de $\mathbb{F}_{p^2}$ equivale a 16 palabras de 32 bits. Esto implica que por cada multiplicación en $\mathbb{F}_{p^2}$ se deben transferir 48 palabras, 32 de ellas durante el envío de los operandos desde el microprocesador y otras 16 correspondientes al resultado devuelto por el coprocesador. Así, para la multiplicación dispersa en $\mathbb{F}_{p^{12}}$, que con trece operaciones simultáneas de multiplicación en $\mathbb{F}_{p^2}$ es la que impone el requisito más exigente, se deben realizar un total de 624 trasferencias de 32 bits entre el microprocesador y el coprocesador. La alternativa adecuada para gestionar este volumen de transferencias de manera eficiente en el contexto de la plataforma Zynq-7000 consiste en establecer enlaces punto-a-punto mediante interfaces *AXI-Stream* [52] de conjunto con la técnica de Acceso Directo a Memoria. Además, debido a que los operandos se organizan y transfieren formando secuencias de palabras, es oportuno que la estructura referida anteriormente para el procesamiento en forma de *pipeline* sea concebida para que sus diferentes etapas actúen también sobre palabras. De esta manera, a medida que lleguen al coprocesador las palabras de los operandos, se van introduciendo en la cadena de procesamiento, devolviéndose el resultado a partir de que se genere la primera palabra del mismo. Esto permite cierto grado de solapamiento entre la transferencia y el procesamiento de datos, lo cual contribuye a reducir el tiempo total de ejecución.

5.2.- ARQUITECTURA DEL COPROCESADOR

La Figura 1 muestra la estructura del coprocesador de multiplicación en $\mathbb{F}_{p^2}$. Las interfaces de entrada y salida de datos corresponden a enlaces *AXI-Stream* de tipo esclavo y maestro respectivamente. Para la interfaz de configuración se emplea una interfaz esclava de bus AXI dedicada a la recepción de los parámetros de operación del coprocesador. El bloque de procesamiento contiene los aceleradores aritméticos correspondientes a las operaciones de suma, resta, multiplicación y reducción modular de números enteros de múltiple precisión que conforman la cadena de procesamiento de una multiplicación en $\mathbb{F}_{p^2}$. Si bien los enlaces *AXI-Stream* de las interfaces de entrada y salida manipulan un ancho de datos de 32 bits, el mejor balance entre rendimiento y consumo hardware de los aceleradores aritméticos del bloque de procesamiento se obtuvo para un ancho de palabra de 64 bits. Por tal motivo, cada par de transferencias de 32 bits recibidas en la interfaz de entrada de datos se agrupan en palabras de 64 bits que son enviadas hacia la FIFO (*First In-First Out*) de entrada del bloque de procesamiento. Por otro lado, cada vez que se genera una nueva palabra del resultado, esta se escribe en la FIFO de salida, la cual ofrece una interfaz de datos de 64 bits hacia la lógica interna del bloque de procesamiento y otra de 32 bits hacia la interfaz de salida de datos. De esta manera el ajuste del ancho de palabra se produce de forma automática pues cada palabra escrita (64 bits) en la FIFO desencadena la lectura de dos palabras de 32 bits para ser transferidas mediante el enlace *AXI-Stream* de la interfaz de salida.

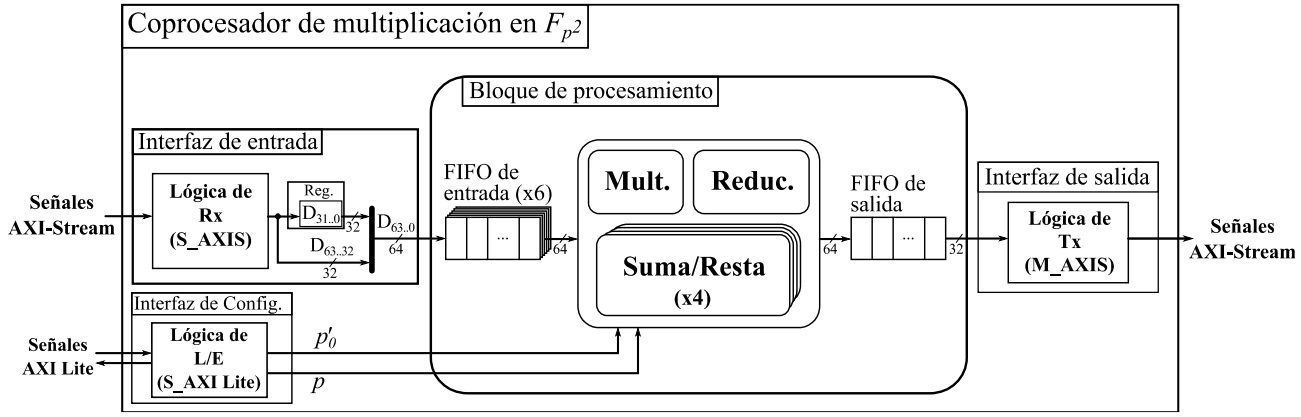

Figura 1

Estructura del coprocesador de multiplicación en $\mathbb{F}_{p^2}$.

La arquitectura detallada del bloque de procesamiento se muestra en la Figura 2. Esta arquitectura constituye una variante de implementación hardware del Algoritmo 4 en la que se han tenido en cuenta las premisas de diseño referidas anteriormente. Como se puede apreciar, la cadena de procesamiento está integrada por cuatro módulos de suma y resta (**Suma/Resta #0...#3**), un módulo de multiplicación (**Mult.**) y otro de reducción modular (**Reduc.**). Además, se puede observar un conjunto de memorias de tipo FIFO distribuidas a lo largo de la cadena de procesamiento. Estas tienen como objetivo ir acumulando los datos de entrada de cada uno de los módulos funcionales para acoplar así las diferentes capacidades de procesamiento que estos poseen. De esta forma se evita que un módulo tenga que detener su funcionamiento en espera de que el siguiente se encuentre disponible para admitir nuevos datos de entrada. En particular, las memorias de la **FIFO#0** a la **FIFO#3**, así como la **FIFO#0P** y la **FIFO#1P**, conforman el subsistema de almacenamiento de entrada del bloque de procesamiento donde se alojan los operandos recibidos mediante la interfaz de entrada del coprocesador. Por su parte, en la memoria **FIFO#12** se van almacenando los resultados que son accedidos posteriormente por la interfaz de salida. Las conexiones detalladas tanto de las señales de reloj como de las señales de control de las memorias FIFO se han omitido para mantener la simplicidad del esquema. El módulo identificado como **Lógica de Control** representa la implementación de una máquina de estados finitos que se encarga de gestionar el proceso de ejecución y ajustar el flujo de datos de acuerdo a los requisitos de la cadena de procesamiento.

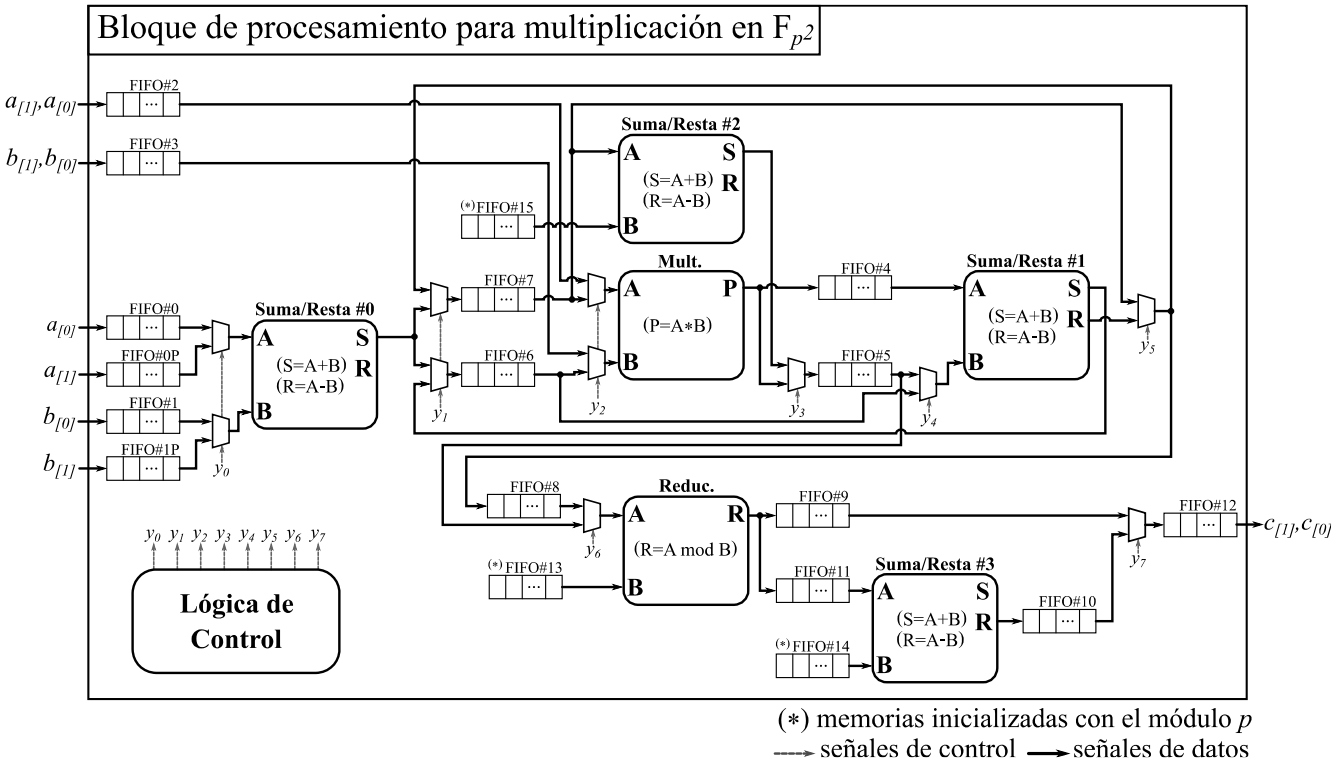

Figura 2

Arquitectura del bloque de procesamiento del coprocesador de multiplicación en $\mathbb{F}_{p^2}$.

A medida que se reciben las palabras de los coeficientes $a_{[0]}, a_{[1]}, b_{[0]}, b_{[1]} \in \mathbb{F}_p$ de los operandos $a = a_{[0]} + a_{[1]}u \in \mathbb{F}_{p^2}$ y $b = b_{[0]} + b_{[1]}u \in \mathbb{F}_{p^2}$, estas se replican convenientemente en varias de las memorias FIFO del subsistema de entrada del bloque de procesamiento. Así, los coeficientes $a_{[0]}$ y $a_{[1]}$ se almacenan en ese orden en la **FIFO#2**, pero también se guardan $a_{[0]}$ en la **FIFO#0** y $a_{[1]}$ en la **FIFO#0P**. De forma análoga, los coeficientes $b_{[0]}$ y $b_{[1]}$ son alojados en ese orden en la **FIFO#3**. El coeficiente $b_{[0]}$ se almacena además en la **FIFO#1**, mientras que $b_{[1]}$ se guarda también en la **FIFO#1P**. Esto posibilita que dichos coeficientes puedan ser leídos de manera independiente por los módulos **Suma/Resta #0** y **Mult**. para efectuar de forma concurrente en una primera etapa de procesamiento las operaciones involucradas en los pasos del 1 al 4 del Algoritmo 4. A medida que se obtienen los resultados intermedios de esta primera etapa, se activa una segunda fase de procesamiento en la que intervienen los módulos **Mult.**, **Suma/Resta #1** y **Suma/Resta #2** para simultanear el cálculo de las operaciones indicadas en los pasos del 5 al 8. Finalmente, como parte de una tercera y última etapa de procesamiento, los módulos **Reduc.** y **Suma/Resta #3** se encargan de efectuar las reducciones modulares de los pasos 9 y 10. Como resultado, estas tres etapas de procesamiento derivan en la conformación de una estructura de *pipeline* que satisface los propósitos de las dos primeras premisas de diseño definidas al inicio de este apartado.

La Figura 3 muestra el flujo de datos detallado de la estructura de *pipeline* del coprocesador de multiplicación en $\mathbb{F}_{p^2}$. De esta forma, se pueden observar la secuencia en que ocurren las diferentes operaciones, así como la cantidad de ciclos de reloj que se invierte en cada una de ellas.

Como se puede apreciar, al inicio de cada línea se identifica la operación concreta de que se trata, mientras que dentro de la barra que representa su duración se muestra también la operación, pero indicando en este caso las memorias FIFO (F_i) que se utilizan como fuente y destino de datos. Además, al final de cada línea se puede observar qué módulo de aceleración aritmética se encuentra asociado a la ejecución de la operación. Aunque también se resaltan las tres etapas de procesamiento de la estructura de *pipeline*, nótese que los límites entre una y otra se encuentran solapados. Esto se debe a que los módulos aritméticos han sido implementados para operar palabra a palabra siguiendo su propio flujo interno de *pipeline*. Por tal motivo, basta que se encuentre lista la primera palabra del resultado de la última operación de una etapa para que se pueda iniciar la primera operación de la etapa de procesamiento siguiente. El hecho de que los módulos de aceleración aritmética cuenten con una estructura de *pipeline* interna también incide en la reducción del tiempo de ejecución de cada etapa y por ende del tiempo

de ejecución total, lo cual ha permitido que el bloque de procesamiento invierta solamente 82 ciclos de reloj para ejecutar una multiplicación en $\mathbb{F}_{p^2}$. Además, como se puede apreciar, a partir de los 28 ciclos ya es posible iniciar otra multiplicación. Así, mientras existan operandos en el subsistema de almacenamiento (memorias FIFO) de entrada, el bloque de procesamiento se mantiene ejecutando operaciones de multiplicación en $\mathbb{F}_{p^2}$ generando cada nuevo resultado en intervalos de 28 ciclos de reloj luego de que hayan transcurrido los 82 ciclos correspondientes a la primera multiplicación. Esto se traduce, por ejemplo, en que las trece multiplicaciones en $\mathbb{F}_{p^2}$ involucradas en una operación de multiplicación dispersa en $\mathbb{F}_{p^{12}}$ tarden unos 418 ciclos en ejecutarse. Esto representa aproximadamente un 60% de ahorro respecto a una variante que no explotase el *pipeline*.

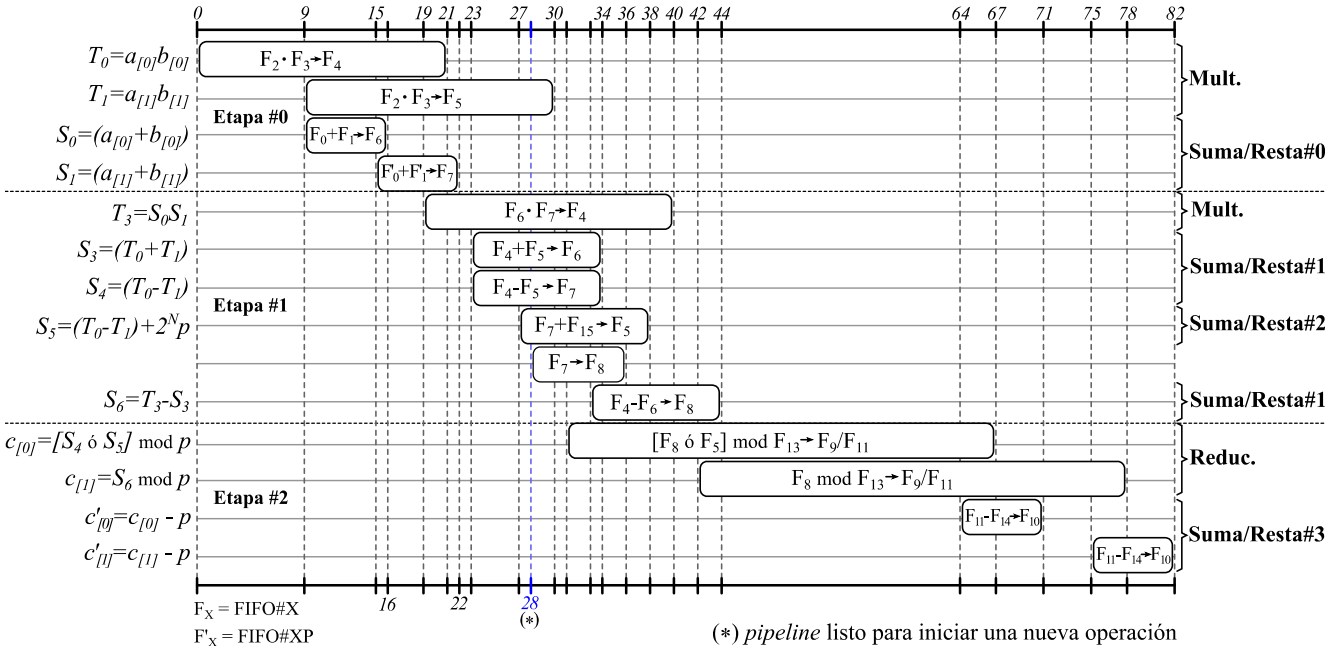

Figura 3

Flujo de *pipeline* del coprocesador de multiplicación en $\mathbb{F}_{p^2}$.

6.- DISCUSIÓN DE RESULTADOS

La implementación del coprocesador de multiplicación en $\mathbb{F}_{p^2}$, así como la construcción del sistema de procesamiento híbrido hardware/software desplegado en el dispositivo Zynq-7000 XC7Z020 para el cálculo del emparejamiento AteOpt-BN fueron realizadas en el entorno de desarrollo *Vivado Design Suite v2016.4*. Para la descripción del coprocesador se utilizó el lenguaje VHDL (*Very-High-Speed-Integrated-Circuit Hardware Description Language*). Tras comprobar el correcto funcionamiento de cada uno de sus componentes, el coprocesador fue empaquetado como módulo IP (*Intellectual Property*) y añadido al sistema de procesamiento mostrado en la Figura 4. Como se puede apreciar la estructura del sistema se completa con el bloque ZYNQ, un controlador de acceso directo a memoria AXI_DMA, los componentes de interconexión *axi_mem_intercon*, *axi_periph* y *xlconcat*, así como el bloque de generación de *reset*.

El bloque ZYNQ representa el sistema de procesamiento ARM Cortex-A9 presente en la familia de dispositivos SoC-FPGA Zynq-7000 al cual se le han habilitado su interfaz AXI de altas prestaciones (S_AXI_ACP) y las entradas de interrupción provenientes de la sección de lógica programable (IRQ_F2P). A la interfaz maestra M_AXI_GP0 se han acoplado las interfaces esclavas S_AXI y S_AXI_LITE del coprocesador de multiplicación en $\mathbb{F}_{p^2}$ y el controlador AXI_DMA respectivamente. A través de estas el procesador ARM configura los parámetros de uno y otro componente. La frecuencia del procesador quedó configurada a 667 MHz mientras que para la lógica programable se utiliza un reloj de 100 MHz.

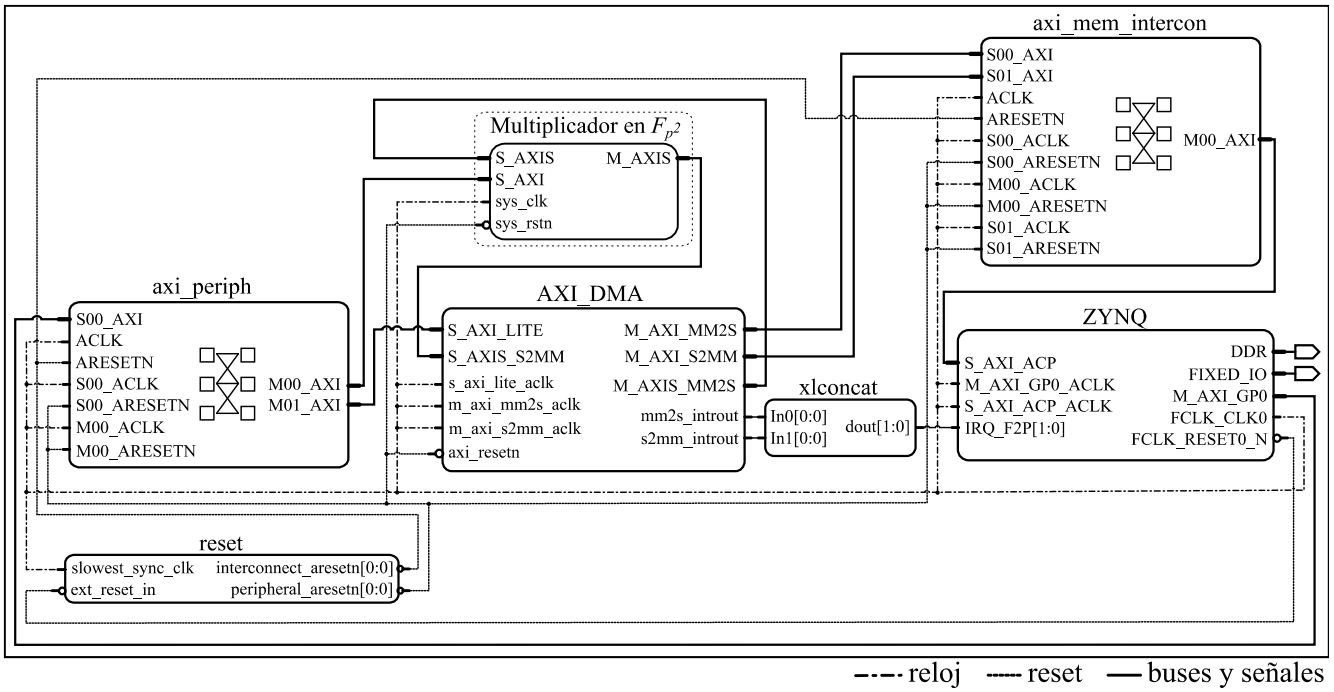

Figura 4

Sistema de procesamiento para la implementación híbrida hardware/software del emparejamiento AteOpt-BN utilizando el coprocesador de multiplicación en $\mathbb{F}_{p^2}$.

El controlador AXI_DMA fue configurado en modo simple. Las interfaces maestras de bus AXI de los canales de lectura (M_AXI_MM2S) y escritura (M_AXI_S2MM) del controlador de DMA han sido acopladas a la interfaz S_AXI_ACP del bloque ZYNQ para garantizar el acceso hacia el subsistema de memoria del procesador ARM. Las interfaces *AXI-Stream* M_AXIS_MM2S y S_AXIS_S2MM se conectan directamente a los bloques de entrada y salida de datos del coprocesador. El canal de lectura en memoria del controlador AXI_DMA fue configurado para realizar ráfagas de hasta 256 transferencias de 32 bits, valor máximo admitido [53]. Esto permite transferir ininterrumpidamente hasta 16 elementos de $\mathbb{F}_{p^2}$ (operandos de ocho multiplicaciones) desde el procesador ARM hacia el coprocesador. En cambio, el canal de escritura en memoria quedó configurado para realizar ráfagas de 8 transferencias de 32 bits como máximo, lo cual coincide con el tamaño de un elemento en $\mathbb{F}_p$. De esta forma, independientemente del total de multiplicaciones a realizar, a partir de que el controlador AXI_DMA reciba el primer coeficiente en $\mathbb{F}_p$ correspondiente al primer resultado de una multiplicación en $\mathbb{F}_{p^2}$ se comenzarán a escribir estos coeficientes en memoria. Así, al solaparse el flujo de datos desde el coprocesador hacia el controlador AXI_DMA con el envío de los mismos desde el controlador AXI_DMA hacia la memoria del procesador ARM, se obtiene un impacto positivo en el rendimiento del mecanismo de transferencias.

Los resultados de implementación del coprocesador de multiplicación en $\mathbb{F}_{p^2}$ exhiben porcentajes de utilización en el dispositivo Zynq XC7Z020 de un 24% (3186) de *slices*, un 5% (7) de bloques dedicados de RAM y un 76% (168) de bloques DSP, siendo este último el elemento crítico del diseño en cuanto a consumo de recursos hardware. Este volumen de ocupación de bloques DSP está asociado a la implementación de las estructuras de *pipeline* de los módulos de multiplicación y reducción modular de números enteros. Para obtener el mayor rendimiento posible, se utilizan en cada caso cuatro unidades de procesamiento que operan en paralelo. Cada una de estas unidades emplean 16 bloques DSP en el caso de la multiplicación y 26 bloques DSP en el caso de la reducción modular, lo cual conduce al elevado consumo de este tipo de recurso. De no haber implementado estos módulos en base a estructuras de *pipeline*, la utilización de bloques DSP del coprocesador se reduciría a tan sólo 42. No obstante, el flujo de procesamiento mostrado en la Figura 3 sería impactado desfavorablemente ya que las operaciones de multiplicación y reducción modular no pudieran ocurrir de forma solapada. Esto implicaría que una multiplicación en $\mathbb{F}_{p^2}$ tardaría alrededor de 130 ciclos de reloj, retrasándose la posibilidad de iniciar cada nueva operación hasta 63 ciclos después de iniciada la anterior. En tal caso, por ejemplo, para ejecutar las trece multiplicaciones en $\mathbb{F}_{p^2}$

requeridas en una multiplicación dispersa en $\mathbb{F}_{p^{12}}$, se invertirían unos 886 ciclos de reloj, lo que representa un 52.8% más que los 418 ciclos que invierte la propuesta de solución basada en estructuras de *pipeline*.

La parte software del diseño fue desarrollada en la herramienta *Xilinx SDK v2016.4*. Para la interacción con el coprocesador se implementaron seis funciones (`cpDualFp2Mult`, `cpThreeFp2Mult`, `cpQuadFp2Mult`, `cpFifthFp2Mult`, `cpSixthFp2Mult` y `cpThirteenthFp2Mult`) encargadas de la ejecución en ráfaga de dos, tres, cuatro, cinco, seis y trece multiplicaciones en cada caso. Estas funciones han sido empleadas en los diferentes niveles de procesamiento aritmético que requieren realizar operaciones de multiplicación en $\mathbb{F}_{p^2}$ para completar el cálculo de un emparejamiento.

Para evaluar el desempeño del coprocesador primero se estudió el rendimiento de cada de las funciones anteriores. A partir de medir el tiempo de ejecución de 10000 repeticiones independientes de cada una de ellas, se calcularon los ciclos de reloj promedio que tardan en ejecutarse. Esto incluye el tiempo que se invierte en transferencias de datos entre el procesador ARM y el coprocesador. Los resultados correspondientes se pueden observar en la Tabla 1, los cuales, como referencia, se contrastan con los tiempos calculados en cada caso para una cantidad equivalente de multiplicaciones partiendo de las métricas presentadas en [15] para dos multiplicaciones en $\mathbb{F}_{p^2}$ simultáneas utilizando procesamiento doble núcleo. Como se puede apreciar, los incrementos de rendimiento al utilizar el coprocesador implementado en este trabajo respecto a la solución software basada en procesamiento doble núcleo de [15] van desde un 19% para dos multiplicaciones hasta un 55.8% para el caso de trece multiplicaciones.

Tabla 1
Tiempos de ejecución de las funciones de multiplicación en $\mathbb{F}_{p^2}$ con el coprocesador

Cantidad de multiplicaciones en $\mathbb{F}_{p^2}$	Tiempos de ejecución (10^3 ciclos de reloj)†	
	Procesamiento doble núcleo [15]	Coprocesador (este trabajo)
2	2.47	2.38
3	4.17	2.79
4	4.94	3.16
5	6.64	3.54
6	7.41	3.96
13	16.5	7.29

† Los ciclos de reloj se expresan en base a la señal de reloj de 667 MHz del procesador ARM.

La Tabla 2 muestra el impacto favorable que tiene el empleo del coprocesador de multiplicación en $\mathbb{F}_{p^2}$ en una implementación híbrida hardware/software para incrementar el rendimiento del emparejamiento AteOpt-BN. Los tiempos para el bucle de Miller, la etapa de exponenciación final y por ende el tiempo total de ejecución del emparejamiento exhiben mejoras significativas en la solución propuesta en este trabajo en comparación con los valores obtenidos en [15] para la implementación basada en procesamiento doble núcleo. De igual modo se evidencia una mejora notable respecto a la solución híbrida hardware/software reportada en [20], única solución previa de este tipo identificada en el estado del arte para la implementación de emparejamientos bilineales. Concretamente, se han logrado mejoras en el tiempo de cálculo del emparejamiento AteOpt-BN de un 32% respecto a la solución de procesamiento doble núcleo de [15] y de un 22.5% en comparación con los resultados de la implementación híbrida de [20].

Aunque en [20] los autores también incluyen soporte hardware para las operaciones de suma, resta y cuadrado en $\mathbb{F}_{p^2}$, su solución emplea una arquitectura con ancho de palabra de 32 bits. Incluso, en el caso de la multiplicación y el cuadrado, las operaciones básicas se realizan a partir de una unidad de multiplicación y acumulación de 32×16 bits. Esto contrasta con la arquitectura del coprocesador propuesto en este trabajo donde las unidades de procesamiento subyacentes emplean un ancho de palabra de 64 bits. Esto conduce a una reducción de la cantidad de ciclos de reloj que se requieren para completar cada operación, lo cual se traduce en un mejor rendimiento del proceso de cálculo de un emparejamiento en cuanto a tiempo de ejecución.

Tabla 2
Comparación de los tiempos de ejecución del emparejamiento AteOpt-BN

Implementación	Tiempos de ejecución (10^3 ciclos de reloj)		
	Bucle de Miller	Exponenciación final	AteOpt-BN
Procesamiento doble núcleo [15]	4375	4456	8849
Híbrida hardware/software [20]	–	–	7763
Híbrida hardware/software (este trabajo)	2917	3085	6017

7.- CONCLUSIONES

Las operaciones aritméticas involucradas en el cálculo de un emparejamiento bilineal exhiben un alto grado de paralelismo. Debido a ello, resulta imprescindible explotar aquellas características tecnológicas de las plataformas de implementación que permitan aprovechar este paralelismo en pos de obtener implementaciones eficientes. En este sentido, el presente trabajo ha abordado la implementación de un coprocesador hardware para acelerar las operaciones de multiplicación en $\mathbb{F}_{p^2}$, por ser las de mayor incidencia en el cálculo de un emparejamiento. La arquitectura propuesta explota el uso de estructuras de *pipeline* en varios niveles, lo cual hace posible acelerar una multiplicación en sí y además simultanear el cálculo de varias operaciones de multiplicación. El impacto favorable de emplear este coprocesador se ha comprobado en una implementación híbrida hardware/software sobre un SoC-FPGA Zynq-7000 del emparejamiento Ate óptimo en curvas de Barreto-Naehrig que logra mejorar los resultados de trabajos previos que emplean estrategias de aceleración similares. Por otra parte, si bien se ha tomado como ejemplo el emparejamiento AteOpt-BN, el hecho de que las operaciones de alto nivel se implementen en software le otorga un nivel de flexibilidad a la solución propuesta que permite extrapolar los resultados obtenidos en este trabajo a la implementación de otros tipos de emparejamientos al aprovechar el paralelismo y la aceleración hardware que proporciona el coprocesador de multiplicación en $\mathbb{F}_{p^2}$.

AGRADECIMIENTOS

Esta investigación ha sido financiada parcialmente por el Consejo Superior de Investigaciones Científicas (CSIC) de España a través del proyecto COOPA 20141 correspondiente al Programa de Cooperación Científica para el Desarrollo i-COOP+.

REFERENCIAS

1. Joux A. A One Round Protocol for Tripartite Diffie-Hellman. In th International Symposium on Algorithmic Number Theory; 2000. p. 385-393.
2. Sakai R, Ohgishi K, and Kasahara M. Cryptosystems Based on Pairings. In 2000 Symposium on Cryptography and Information Security - SCIS 2000; 2000.
3. Boneh D, Franklin M. Identity-Based Encryption from the Weil Pairing. In Advances in Cryptology - CRYPTO 2001; 2001. p. 213-229.
4. Sakai R, Kasahara M. ID based Cryptosystems with Pairing on Elliptic Curve. IACR Eprint archive. 2003.
5. Boneh D, Boyen X. Efficient Selective-ID Secure Identity-Based Encryption Without Random Oracles. In Advances in Cryptology - EUROCRYPT 2004; 2004. p. 223-238.
6. Boneh D, Lynn B, Shacham H. Short signatures from the Weil pairing. In Advances in Cryptology - ASIACRYPT 2001; 2001. p. 514-532.
7. Waters B. Ciphertext-Policy Attribute-Based Encryption: An Expressive, Efficient, and Provably Secure Realization. In Public Key Cryptography - PKC 2011; 2011. p. 53-70.
8. Herbert V, Biswas B, Fontaine C. Design and implementation of low-depth pairing-based homomorphic encryption scheme. Journal of Cryptographic Engineering. 2019; 9(2): 185-201.

9. Ghosh S, Verbauwhede I, and Roychowdhury D. Core Based Architecture to Speed Up Optimal Ate Pairing on FPGA Platform. In Pairing-Based Cryptography - Pairing 2012 ; 2013.
10. Liu J, Cheng D, Guan Z, and Wang Z. A High Speed VLSI Implementation of 256-bit Scalar Point Multiplier for ECC over GF(p). In 2018 IEEE International Conference on Intelligence and Safety for Robotics; 2018. p. 184-191.
11. Grewal G, Azarderakhsh R, Longa P, Hu S, Jao D. Efficient Implementation of Bilinear Pairings on ARM Processors. In Selected Areas in Cryptography; 2012. p. 149-165.
12. Sánchez AH, Rodríguez-Henríquez F. NEON Implementation of an Attribute-Based Encryption Scheme. In Applied Cryptography and Network Security; 2013. p. 322-338.
13. Verma R. Efficient Implementations of Pairing-Based Cryptography on Embedded Systems. Rochester Institute of Technology; 2015.
14. Cuiman R, Cabrera AJ, Sánchez-Solano S. Speeding up elliptic curve arithmetic on ARM processors using NEON instructions. Revista de Ingeniería Electrónica, Automática y Comunicaciones. 2020; 41(3): 1-20.
15. Cuiman R, Cabrera AJ, and Sánchez-Solano S. Implementing Cryptographic Pairings on ARM Dual-Core Processors. IEEE Latin America Transactions. 2020; 18(2): 232-240.
16. Kammler D, Zhang D, Schwabe P, Scharwaechter H, Langenberg M, Auras D, et al. Designing an ASIP for Cryptographic Pairings over Barreto-Naehrig Curves. In Cryptographic Hardware and Embedded Systems - CHES 2009; 2009. p. 254-271.
17. Cheung R, Duquesne S, Fan J, Guillermin N, Verbauwhede I, and Yao GX. FPGA Implementation of Pairings using Residue Number System and Lazy Reduction. In Cryptographic Hardware and Embedded Systems - CHES 2011 ; 2011.
18. Fan J, Vercauteren F, Verbauwhede I. Efficient Hardware Implementation of Fp-arithmetic for Pairing-Friendly Curves. IEEE Transactions on Computers. 2012; 61(5): 676-685.
19. Ghosh S, Mukhopadhyay D, Roychowdhury D. "Secure Dual-Core Cryptoprocessor for Pairings Over Barreto-Naehrig Curves on FPGA Platform. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 2013; 21(3): 434-442.
20. Unterluggauer T, Wenger E. Efficient Pairings and ECC for Embedded Systems. In Cryptographic Hardware and Embedded Systems - CHES 2014; 2014. p. 298-315.
21. Sghaier A, Loubna G, Zeghid M, Duquesne S, Machhout M. Area-Efficient Hardware Implementation of the Optimal Ate Pairing over BN curves. Cryptology ePrint Archive. 2015;: 1-21.
22. Bahadori M, Kimmo J. Compact and Programmable yet High-Performance SoC Architecture for Cryptographic Pairings. In 2020 30th International Conference on Field-Programmable Logic and Applications; 2020. p. 176-184.
23. Bag A, Basu Roy D, Patranabis S, Mukhopadhyay D. FlexiPair: An Automated Programmable Framework for Pairing Cryptosystems. IEEE Transactions on Computers. 2021.
24. Vercauteren F. Optimal Pairings. IEEE Transactions on Information Theory. 2010; 56(1): 455-461.
25. Barreto PSLM, Naehrig M. Pairing-Friendly Elliptic Curves of Prime Order. In Preneel B, Tavares S, editors. Selected Areas in Cryptography - SAC 2005, LNCS 3897; 2006: Springer-Verlag. p. 319-331.
26. Sakemi Y, KT, Saito T, Wahby RS. Pairing-Friendly Curves. Internet Engineering Task Force; 2020.
27. Freeman D, SM, Teske E. A Taxonomy of Pairing-Friendly Elliptic Curves. Journal of Cryptology. 2010; 23(2): 224-280.
28. Duquesne S, El Mrabet N, Haloui S, Rondepierre F. Choosing and generating parameters for pairing implementation on BN curves. Applicable Algebra in Engineering, Communication and Computing. 2018; 29(2): 113-147.
29. Nogami Y, Akane M, SY, Kato H, Morikawa Y. Integer Variable Chi - Based Ate Pairing. In Pairing-Based Cryptography - Pairing 2008; 2008. p. 178-191.
30. Naehrig M. Constructive and Computational Aspects of Cryptographic Pairings. Eindhoven University of Technology; 2009.
31. Miller VS. Short Programs for functions on Curves. ; 1986.
32. Miller VS. The Weil Pairing, and Its Efficient Calculation. Journal of Cryptology. 2004; 17(4): 235-261.
33. Scott M, Benger N, Charlemagne M, Dominguez LJ, Kachisa EJ. On the final exponentiation for calculating pairings on ordinary elliptic curves. In Pairing-Based Cryptography - Pairing 2009 ; 2009. p. 78-88.
34. Scott M, Barreto PSLM. Compressed Pairings. In Advances in Cryptology - CRYPTO 2004 ; 2004. p. 140-156.

35. Olivos J. On Vectorial Addition Chains. Journal of Algorithms. 1981; 2(1): 13-21.
36. Bernstein DJ, Lange T. Analysis and optimization of elliptic-curve single-scalar multiplication. In Mullen GLea, editor. Finite fields and applications: Eighth international conference on finite fields and applications; 2007; Melbourne.
37. Silverman JH. The Arithmetic of Elliptic Curves. Segunda ed. New York: Springer; 2009.
38. Chudnovsky DV, Chudnovsky GV. Sequences of numbers generated by addition in formal groups and new primality and factorization tests. Advances in Applied Mathematics. 1986; 7(4): 385-434.
39. Costello C, Lange T, Naehrig M. Faster Pairing Computations on Curves with High-Degree Twists. In Public Key Cryptography - PKC 2010 ; 2010. p. 224-242.
40. Koblitz N, Menezes A. Pairing-based cryptography at high security levels. In Cryptography and Coding ; 2005. p. 13-36.
41. Benger N, Scott M. Constructing Tower Extensions of Finite Fields for Implementation of Pairing-Based Cryptography. In Arithmetic of Finite Fields vol. 6087, (Istambul: Springer Berlin Heidelberg, 2010), pp. 180--195.; 2010. p. 180-195.
42. Miyaji A, NM, Takano S. New Explicit Conditions of Elliptic Curve Traces for FR Reduction. IEICE TRANSACTIONS on Fundamentals of Electronics, Communications and Computer Sciences. 2001; E84-A(5): 1234-1243.
43. Barreto PSLM, Lynn B, Scott M. Constructing Elliptic Curves with Prescribed Embedding Degrees. In Security in Communication Networks; 2003. p. 257-267.
44. Kachisa EJ, Schaefer EF, Scott M. Constructing Brezing-Weng pairing friendly elliptic curves using elements in the cyclotomic field. In Pairing-Based Cryptography - Pairing 2008 ; 2008. p. 126-135.
45. Montgomery PL. Modular Multiplication without Trial Division. Mathematics of Computation. 1985; 44(170): 519-521.
46. Shoup V. A Computational Introduction to Number Theory and Algebrac. 2nd ed. New York: Cambridge University Press; 2009.
47. Karatsuba AA, Ofman Y. Multiplication of Multidigit Numbers on Automata. Soviet Physics Doklady. 1963; 7(7): 595-596.
48. Devegili AJ, O hEigeartaigh C, Scott M, Dahab R. Multiplication and Squaring on Pairing-Friendly Fields. IACR Eprint archive. 2006.
49. Lim CH, Hwang HS. Fast Implementation of Elliptic Curve Arithmetic in GF(p^n). In ; 2000. p. 405-421.
50. Avnet, Inc. ZedBoard (Zynq Evaluation and Development) Hardware User's Guide. 2014..
51. Crockett LH, Elliot RA, Enderwitz MA, Stewart RW. The Zynq Book: Embedded Processing with the ARM Cortex-A9 on the Xilinx Zynq-7000 All Programmable SoC. Primera ed.: Strathclyde Academic Media; 2014.
52. ARM, Inc. AMBA 4 AXI4-Stream Protocol Specification. 2010..
53. Xilinx, Inc. AXI DMA v7.1 LogiCORE IP Product Guide. 2019..

CONFLICTO DE INTERESES

Ninguno de los autores ha manifestado la existencia de posibles conflictos de intereses que debieran ser declarados en relación con este artículo.

CONTRIBUCIONES DE LOS AUTORES

Raudel Cuiman Márquez: conceptualización, investigación, metodología, curación de datos, software, validación – verificación, supervisión, redacción – revisión y edición.

Alejandro José Cabrera Sarmiento: conceptualización, investigación, supervisión, redacción – revisión y edición.

Santiago Sánchez Solano: conceptualización, investigación, supervisión, redacción – revisión y edición.

AUTORES

Raudel Cuiman Márquez, Ingeniero en Automática en 2009, Doctor en Ciencias Técnicas en 2022 por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Auxiliar del Departamento de Automática y Computación de la CUJAE, La Habana, Cuba. ORCID: 0000-0002-5145-7612. Email: raudel@automatica.cujae.edu.cu. Sus intereses de investigación están relacionados con la implementación eficiente de algoritmos criptográficos sobre sistemas empotrados.

Alejandro J. Cabrera Sarmiento, Ingeniero Electricista, Doctor en Ciencias Técnicas por la Universidad Tecnológica de La Habana "José Antonio Echeverría" (CUJAE), Profesor Titular del Departamento de Automática y Computación de la CUJAE, La Habana, Cuba. ORCID: 0000-0003-4129-911X. Email: alex@automatica.cujae.edu.cu. Sus líneas de investigación principales están relacionadas con el desarrollo de sistemas empotrados basados en FPGA y la aceleración de algoritmos sobre hardware reconfigurable, con aplicaciones en procesado de imágenes, control inteligente y criptografía.

Santiago Sánchez-Solano, Doctor en Ciencias Físicas por la Universidad de Sevilla en 1990, Investigador Científico del CSIC adscrito al Instituto de Microelectrónica de Sevilla, IMSE-CNM, (CSIC/Universidad de Sevilla), Sevilla, España. ORCID: 0000-0002-0700-0447. Email: santiago@imse-cnm.csic.es. Sus líneas de investigación se centran en el desarrollo de sistemas empotrados con componentes hardware-software sobre FPGA y la realización microelectrónica de sistemas neurodifusos, así como sus aplicaciones en robótica, procesamiento de imágenes, seguridad y redes de sensores inteligentes.