Multiplicación Escalar en Curvas de Koblitz: Arquitectura en Hardware Reconfigurable

Juan Manuel Cruz Alcaraz, Francisco Rodríguez-Henríquez
Sección de Computación, Departamento de Ingeniería Eléctrica
Centro de Investigación y de Estudios Avanzados del IPN
Av. Instituto Politécnico Nacional No. 2508, México D.F.
Correo electrónico: jmcruz@computacion.cs.cinvestav.mx, francisco@cs.cinvestav.mx

Resumen

Se presenta el diseño de una arquitectura en hardware reconfigurable capaz de calcular la operación de multiplicación escalar kP en curvas elípticas de Koblitz. La arquitectura consiste de dos unidades independientes de cómputo que trabajan bajo el esquema productor-consumidor. Empleando aritmética entera, la primera unidad produce la expansión $\omega\tau$ NAF del escalar k; mientras que la segunda unidad usa aritmética de campos finitos binarios para calcular a partir de la expansión generada por la primera etapa, la multiplicación escalar kP. Se reportan los costos en recursos de hardware así como los desempeños en tiempo obtenidos al implementar la arquitectura en un dispositivo FPGA Virtex 2 XC2V4000. Finalmente, se incluye una tabla en la que se compara el desempeño obtenido por nuestro diseño contra otros trabajos similares publicados en la literatura especializada.

1. Introducción

Los algoritmos criptográficos de llave pública están basados en aritmética definida sobre grupos abelianos multiplicativos y aditivos, como los que se encuentran en campos finitos primos y binarios.

En 1985, Victor Miller y N. Koblitz propusieron un criptosistema de llave pública análogo al esquema de El Gamal en el cual el campo finito binario GF(2^m) es substituido por el grupo de puntos en una curva elíptica definida sobre un campo finito. La operación básica de este criptosistema es la multiplicación escalar, la cual está definida como el cálculo del múltiplo entero k de un punto fio P de la curva elíptica, esto es, el cálculo de la operación $kP = \underbrace{P + P + \cdots + P}_{kveces}$.

Posteriormente, Koblitz introdujo una familia de curvas que admiten multiplicaciones escalares especialmente veloces, la cual reemplaza las operaciones de doblado de puntos por operaciones cuadráticas. En este trabajo se presenta una implementación en hardware reconfigurable de los algoritmos presentados en [1, 2]. Se propone una arquitectura de multiplicación compuesta por dos unidades aritméticas independientes (campos finitos binarios y aritmética entera). Un bloque de memoria compartida es usado para comunicar resultados intermedios. Se presenta su implementación en VHDL y su respectiva síntesis sobre un dispositivo FPGA Xilinx XC2V4000 así como resultados en tiempos de ejecución de una multiplicación escalar y recursos utilizados del dispositivo.

El resto de este artículo está organizado como sigue. En la Sección 2 se presenta un resumen de los principales conceptos matemáticos relacionados con el cálculo de multiplicaciones escalares en curvas elípticas de Koblitz. En seguida en la Sección 3 se describen los algoritmos asociados a la operación de multiplicación escalar. En la Sección 4 se describe la arquitectura propuesta en este trabajo y se discuten los criterios de diseño para lograr su implementación

en dispositivos de hardware reconfigurable. Posteriormente, en la Sección 5, se describe brevemente las operaciones principales incluidas en la unidad de aritmética de campos finitos binarios. En seguida, en la Sección 6 se reportan los resultados experimentales obtenidos. Nuestras conclusiones y trabajo futuro son detallados en la Sección 7.

2. Preliminares Matemáticos

2.1. Curvas elípticas: definición y ley de grupo

Se dice que una curva elíptica no singular sobre un campo finito binario $E(GF[2^m])$ está definida como el conjunto de puntos $(x, y) \in GF[2^m] \times GF[2^m]$ que satisfacen la siguiente ecuación afín:

$$y^2 + xy = x^3 + ax^2 + b \tag{1}$$

Donde a y $b \in GF[2^m]$, junto con el punto al infinito denotado por O. Es posible construir un grupo abeliano aditivo a partir del conjunto de puntos y la definición de una operación de adición entre puntos de la curva (1), la cual debe cumplir las leyes conmutativa, asociativa, inverso aditivo, elemento identidad y cerradura. La suma de dos puntos en la curva elíptica, conocida como su *ley de grupo*, se define como sigue.

Sean $P = (x_1, y_1)$ y $Q = (x_2, y_2)$ dos puntos que pertenecen a la curva definida en (1). Se sabe que $-P = (x_1, x_1 + y_1)$. Asimismo, para todo P en la curva se tiene que, $P + O = O + P = P$. Si $Q \neq -P$, entonces, $P + Q = (x_3, y_3)$, donde:

$$x_3 = \begin{cases} (\frac{y_1+y_2}{x_2 x_2})^2 + \frac{y_1+y_2}{x_1+x_2} + x_1 + x_2 + a & P \neq Q \\ x_1^2, \frac{b}{x_1^2} & P = Q \end{cases} \tag{2}$$

$$y_3 = \begin{cases} (\frac{y_1+y_2}{x_2 x_2})(x_1 + x_3) + x_3 + y_1 & P \neq Q \\ x_1^2 + (x_1 + \frac{y_1}{x_1})x_3 + x_3 & P = Q \end{cases} \tag{3}$$

Desafortunadamente, un cálculo directo de las ecuaciones 2 y 3 implicaría que para obtener la suma de puntos es necesario calcular dos inversiones de campo, la cual es una operación muy costosa desde el punto de vista computacional. Las inversiones de campo pueden ser obviadas mediante el uso de coordenadas proyectivas como se explica en la siguiente subsección.

2.2. Coordenadas proyectivas

Se dice que la clase de equivalencia:

$$(X : Y : Z) = \left\{(\lambda^c X, \lambda^d Y, \lambda Z) : \lambda \in K^*\right\}.$$

es un *punto proyectivo* [3], y (X, Y, Z) un *punto representativo* de dicha clase. Específicamente, si $Z \neq 0$, $(\frac{X}{Z^c}, \frac{Y}{Z^d}, 1)$ es un punto representativo de la clase de equivalencia $(X : Y : Z)$.

La ecuación (1) para una curva elíptica $E(K)$ puede ser definida en coordenadas proyectivas al reemplazar x por $\frac{X}{Z^c}$ y y por $\frac{Y}{Z^d}$.

Los valores de las constantes c y d determinarán las características de la aritmética de curvas elípticas y por lo tanto la definición del algoritmo de adición de puntos en dicha representación.

López y Dahab propusieron un representación en puntos proyectivos para puntos sobre curvas elípticas donde $c = 1$ y $d = 2$ [3]. El punto proyectivo $(X : Y : Z)$, con $Z \neq 0$ corresponde al punto afín $X/Z, Y/Z^2$, mientras que el punto 0 queda representado como $(1 : 0 : 0)$. Por último, el punto negativo de $(X : Y : Z)$ es $(X : X + Y : Z)$. Las fórmulas para calcular la suma (X_3, Y_3, Z_3) de $(X_1 : Y_1 : Z_1)$ y $(X_2 : Y_2 : 1)$ son entonces [3]:

$$A \leftarrow Y_2 \cdot Z_1^2,\ B \leftarrow X_2 \cdot Z_1 + X_1,\ C \leftarrow Z_1 \cdot B,\ C \leftarrow B^2 \cdot (C + aZ_1^2),$$
$$Z_3 \leftarrow C^2,\ E \leftarrow A \cdot C,\ X_3 \leftarrow A^2 + D + E,\ F \leftarrow X_3 + X_2 \cdot Z_3,$$
$$G \leftarrow (X_2 + Y_2) \cdot Z_3^2,\ Y_3 \leftarrow (E + Z_3) \cdot F + G$$

El uso de coordenadas proyectivas permite calcular múltiples adiciones elípticas sin necesidad de computar costosas operaciones de inversiones de campo.

2.3. Curvas de Koblitz

Las curvas de Koblitz se definen como las curvas E_0 y E_1 F_2 tales que [4]:

$$E_a : y^2 + xy = x^3 + ax^2 + 1$$

donde $a \in [0, 1]$. Dichas curvas son atractivas porque exhiben la siguiente propiedad: Si $P = (x, y)$

es un punto en E_a, entonces también el punto (x^2, y^2) está en la curva. Además se cumple que: $(x^4, y^4) + 2(x, y) = \mu(x^2, y^2)$ para cualquier punto (x, y) en E_a, donde $\mu = (-1)^{1-a}$.

Si τ representa la operación de elevar al cuadrado como el mapa de Frobenius sobre F_2, la anterior relación puede ser reescrita como: $\tau(\tau P)+2P = \mu\tau P$.

Resolviendo la ecuación para τ, se puede definir la equivalencia de un mapa de Frobenius con el número complejo $\tau = \frac{\mu+\sqrt{-7}}{2}$.

Cualquier entero positivo k puede ser escrito en forma τNAF como: $k = \sum_{i=0}^{l-1} u_i\tau^i$, donde cada $u_i \in \{0, \pm 1\}$ y l es la longitud de la expansión.

Esta notación describe una función equivalente para calcular kP. En efecto, la multiplicación escalar puede ser calculada con el método NAF de suma-resta. El método NAF de suma-resta calcula kP en un campo finito $GF[2^m]$, utilizando alrededor de m doblados de punto y $m/3$ sumas. El método análogo para la representación τNAF corresponde a evaluar l veces τ Mapeos de Frobenius (esto es, operaciones de campo de elevar al cuadrado), además de $l/3$ sumas de puntos elítpicos.

Sin embargo, es posible procesar ω dígitos al mismo tiempo como sigue. Sea $\omega \geq 2$ un entero positivo. Se define $\alpha_i = i \bmod \tau^w$ para $i \in [1, 3, 5, \ldots, 2^{w-1} - 1]$. Una expansión ω TNAF corresponde a la expresión $\rho = \sum_{i=0}^{l-1} u_i\tau^i$ donde cada $u_i \in [0, \pm\alpha_1, \pm\alpha_3, \ldots, \pm\alpha_{2^{w-1}-1}, u_{l-1}] \neq 0$, y a lo más uno de los dígitos consecutivos es no cero. La longitud de la expansión $\omega\tau$NAF es l.

De esta manera, la expansion $\omega\tau$NAF del escalar k representa la equivalencia entre la multiplicación escalar kP y la expresión: $\alpha_{u_0}P + \tau\alpha_{u_1}P + \tau^2\alpha_{u_2}P + \ldots + \tau^{l-1}\alpha_{u_{l-1}}P$.

Sumarizando, la aritmética de curvas elípticas de Koblitz se basa en sumas elípticas y mapeos τ. El desempeño general del algoritmo está vinculado al desempeño de la aritmética de campos finitos junto con la representación proyectiva de los puntos que permite mejorar la complejidad temporal del algoritmo dado que prácticamente elimina la necesidad de calcular inversiones de campo.

2.4. Algoritmos de Solinas para cálculo de la expansión $\omega\tau$NAF

En este trabajo se diseñó una arquitectura específica para la curva elíptica de Koblitz K-233. Tras hacer un estudio de los algoritmos propuestos por Solinas en [1], se concluye que el tamaño máximo de los operandos en las adiciones en ningún caso es mayor a 235 bits. Así, el algoritmo 1 calcula la expansión $\omega\tau$NAF, usando un método análogo al de los algoritmos NAF generales. Su funcionamiento se basa en la división sucesiva del número $\rho \in \mathbb{Z}[\tau]$ entre τ permitiendo residuos $\pm\alpha_i = \pm i \bmod \tau^\omega$ para $i \in \{1, 3, 5, \ldots, 2^{\omega-1} - 1\}$.

Procedimiento 1 Expansión de coeficientes $\omega\tau$NAF (Véase: [1])

Entrada: : Parámetros de la curva: m, a, s_0, s_1, t_ω. Memoria ROM almacenando $\alpha_u = (\beta, \gamma)$ para $u = 1, 3, \ldots, 2^{\omega-1} - 1$. Escalar reducido: $\rho = (R_0, R_1)$
Salida: $R\tau NAF_\omega(\rho)$

if $R_0 \neq 0$ o $R_1 \neq 0$ **then**
  $A \leftarrow R_1 \cdot t_\omega$
  $U \leftarrow (R_0 \cdot A)$ mods 2^ω, Almacena U
  Direcciona $\alpha_{abs(U)} = (\beta, \gamma)$, $signoU = \{-1, 1\}$, de acuerdo al signo de U.
  $R_0 \leftarrow R_0 - signoU\beta$
  $R_1 \leftarrow R_1 - signoU\gamma$
  $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
while $R_1 \neq -\mu\frac{R_0}{2}$ o $R_0 \neq 0$ **do**
  if R_1 es impar **then**
    $A \leftarrow R_1 \cdot t_\omega$, Almacena $U_{contador}$
    $U \leftarrow (R_0 \cdot A)$ mods 2^ω, Almacena U
    Direcciona $\alpha_{abs(U)} = (\beta, \gamma)$, $signoU = \{-1, 1\}$, de acuerdo al signo de U.
    $R_0 \leftarrow R_0 - signoU\beta$
    $R_1 \leftarrow R_1 - signoU\gamma$, Reinicia $U_{contador}$
    $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$
  else
    Incrementa $U_{contador}$
  $R_0 \leftarrow R_1 + \frac{\mu R_0}{2}$, $R_1 \leftarrow \frac{-R_0}{2}$

Se sabe que la cantidad mínima de ceros consecutivos para una ventana de ω bits es de $\omega - 1$. Cada vez que aparece una secuencia de ceros en la cadena, éstos son contados. La longitud de una secuencia de ceros es almacenada para su posterior uso. De esta manera, el Algoritmo 1 genera una expansión $\omega\tau$NAF con el siguiente formato:

Sean ω_i con $i \in [1, 3, 5, \ldots, 2^{\omega-1} - 1]$ los elementos distintos de cero pertenecientes a la expansión $\omega\tau$NAF. Las secuencias de ceros consecutivos se representan por $z_i \in [\omega - 1, 2w - 2]$. La expansión

$\omega\tau$NAF es entonces obtenida de acuerdo al formato $\omega_0, z_0, \omega_1, z_2, \ldots, z_{i-1}, \omega_i$.

Las condiciones del ciclo *while* en la línea 8 del Algoritmo 1 fueron adaptadas a un esquema de pipeline, de acuerdo a las siguientes observaciones:

1. R_0 será cero si y solo si $R_1 = -\mu\frac{R_0}{2}$.
2. R_1 será cero si y solo si $R_0 = 0$.
3. R_0 será impar si y solo si R_1 es impar, porque $-\mu\frac{R_0}{2}$ es par.

3. Aritmética de Curvas Elípticas

Como se ha explicado, la operación básica en el grupo abeliano formado en curvas elípticas es la adición. Otra operación básica es el doblado de un punto. Esta operación es definida como un caso particular de la suma, la adición de un punto consigo mismo. Sin embargo se define de manera independiente para poder optimizar su implementación.

En este trabajo, los algoritmos propuestos por López y Dahab presentados en [3] han sido paralelizados utilizando funciones atómicas optimizadas para el campo finito $GF(2^{233})$. Se utilizan los registros Qx, Qy y Qz para almacenar un punto elíptico y registros de propósito general Tn.

Procedimiento 2 Suma de puntos en coordenadas LD-afines

Entrada:	$Q = (Q_x : Q_y : Q_z)$ en coordenadas LD , $P = (P_x, P_y)$ en coordenadas afines sobre K-233: $f(z) = z^{233} + z^{74} + 1$ / $GF(2^{233})$
Salida:	$P + Q = (Q_x : Q_y : Q_z)$ en coordenadas LD.
ciclo 1:	**if** $P_z = 0$ **then** regresa $(x_2 : y_2 : 1)$. **else** $Q_x \leftarrow Q_x + Q_z \cdot P_x$.
ciclo 2:	$Q_y \leftarrow Q_y + Q_z^2 \cdot P_y$
ciclo 3:	$T_1 \leftarrow Q_z \cdot Q_x$, $Q_z \leftarrow T_1^2$
ciclo 4:	**if** $Q_x = 0$ **then** **if** $Q_y = 0$ **then** Doblado De Puntos($P_x, P_y, 1$). **else** regresa $[1, 0, 0]$ **else** $T_3 = T_1 \cdot Q_y$.
ciclo 5:	$Q_x = (Q_x^2 \cdot T_1) + (Q_y^2) + T_3$
ciclo 6:	$T_2 = P_x \cdot Q_z + Q_x$
ciclo 7:	$Q_y = (T_3 + Q_z) \cdot T_2$
ciclo 8:	$Q_y + Q_z^2 \cdot (P_x + P_y)$

Procedimiento 3 Doblado de punto es coordenadas LD

Entrada:	$Q = (P_x : P_y : 1)$ en coordenadas LD sobre K-233: $f(z) = z^{233} + z^{74} + 1$ / $GF(2^{233})$
Salida:	$2Q = (Q_x : Q_y : Q_z)$ en coordenadas LD.
ciclo 1:	$Q_z = P_x^2$.
ciclo 2:	$Q_x := Q_z^2 + 1$.
ciclo 3:	$Q_y := Q_x \cdot (P_y^2 + 1) + Q_z$.

Usando los Algoritmos 2 y 3, una suma de puntos elípticos puede ser calculada en 8 ciclos de reloj y un doblado de punto en 3 ciclos de reloj. Un estudio detallado de dichos algoritmos, nos permite identificar 3 funciones atómicas básicas para el diseño de nuestra arquitectura en hardware: $Q_x = (Q_x^2 \cdot T_1) + (Q_y^2) + T_3$; $Q_y = (T_3 + Q_z)$ y $T_1 = Q_z \cdot Q_x$ en paralelo con $Q_z = T_1^2$.

3.1. Multiplicación Escalar

Los algoritmos propuestos por Solinas en [5] y [1] para el cálculo de una multiplicación escalar, una vez que la expansión τNAF del escalar ha sido obtenida, solamente se diferencian por los múltiplos de P utilizados. Ambos algoritmos calculan la multiplicación escalar a partir de la expansión $\omega\tau$NAF aplicando la regla de Horner.

De esta manera, el algoritmo se compone por una sección de precómputo seguida por el cálculo mismo de la expresión de Horner. En ECDSA, el algoritmo generador de firmas digitales siempre realiza la multiplicación escalar en el punto base de la curva elíptica $P \in E(F_q)$. Dado que el desarrollo está enfocado a la generación de firmas, es posible precalcular los correspondientes múltiplos y almacenarlos en memoria.

Como se explicó en la Sección anterior, el Algoritmo 1 provee una expansión del escalar k con el formato: ω_0, z_0, ω_1, z_2, ..., z_{i-1}, donde ω_i representan coeficientes no cero y y $z_i \in [\omega - 1, 2\omega - 2]$. representan la extensión de las rachas de coeficientes cero.

El Algoritmo 4 está diseñado para obtener una implementación eficiente en hardware con la capacidad de calcular secuencias de elevaciones al cuadrado en un solo ciclo de reloj.

Procedimiento 4 Método TNAF con ventana de ω bits para mutiplicación escalar elíptica

Entrada: $\omega\tau NAF(\rho')$ dado como: $\omega_0, z_0, \omega_1, z_2, \ldots, z_{i-1}, \omega_i$ donde $\omega_i \in [1, 3, 5, \ldots, 2^{\omega-1} - 1]$ y $z_i \in [\omega - 1, 2w - 2]$
Algoritmo [5] Almacenamiento no-volátil: $P_u = uP$, for $u \in \{1, 3, 5, ..., 2^{\omega-1} - 1\}$
Algoritmo [1] Almacenamiento no-volátil: $P_u = \alpha_u P$, for $u \in \{1, 3, 5, ..., 2^{\omega-1} - 1\}$
Salida: kP

1: **if** $R_0 \neq 0$ o $R_1 \neq 0$ **then**
2: $Q \leftarrow \vartheta$
3: **for** i from $\frac{2l}{\omega-1} - 1$ downto 0 **do**
4: **if** i es impar **then**
5: $Q \leftarrow \tau^{\omega-1}Q$
6: $\omega\tau NAF_i \leftarrow \omega\tau NAF_i - (\omega - 1)$
7: **if** $\omega\tau NAF_i \neq 0$ **then**
8: $Q \leftarrow \tau^{\omega-1}Q$
9: **else**
10: Direcciona P_u con $\omega\tau NAF_j$
11: $Q \leftarrow \tau Q$
12: **if** $u > 0$ **then**
13: $Q \leftarrow Q + P_u$
14: **else**
15: $Q \leftarrow Q - P_{-u}$

4. Arquitectura de Hardware Reconfigurable

La arquitectura propuesta para el cálculo de una multiplicación escalar elíptica basada en un algoritmo $\omega\tau$NAF corresponde a un esquema productor consumidor.

Se presentan dos unidades de proceso trabajando de forma independiente. La primera unidad de proceso está diseñada para realizar los algoritmos dedicados a realizar la expansión de un factor escalar de 233 bits en coeficientes de una expresión polinomial $\mathbb{Z}[\tau]$ con una ventana de 8 bits.

La segunda unidad de proceso está dedicada a realizar la expresión de Horner correspondiente a la expansión $\omega\tau$NAF y obtener el resultado de la multiplicación escalar. Los Algoritmos 2, 3 y 4 son la base para optimizar dicha unidad de proceso.

El Algoritmo 4 requiere de la expansión $\omega\tau$NAF como entrada para poder calcular la expresión de Horner correspondiente. El Algoritmo 1 genera dicha expansión del factor escalar empezando por el coeficiente menos significativo. Sin embargo el Algoritmo 4 requiere leer el coeficiente más significativo al inicio de su cómputo. Esta dependencia entre ambas etapas impone una restricción importante. La unidad de proceso de curvas elípticas (UPCE) necesita esperar a que la unidad de proceso de expansión τ (UPEτ) genere la expansión por completo.

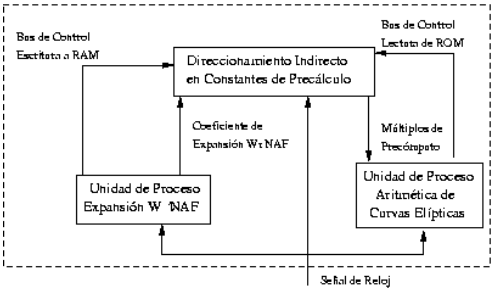

Figura 1: Arquitectura para Multiplicación Escalar

Esto implica que se pueden coordinar mediante un bloque de memoria intermedio dedicado a almacenar la expansión generada por UPEτ para finalmente ser leída en orden inverso por UPCE. La unidad de proceso de curva elípticas no iniciaría su algoritmo hasta que la unidad de proceso de expansión τ lo indique a través de una línea de control.

Los coeficientes de la expansión son almacenados en una memoria RAM de 8x64 bits. Cada coeficiente puede ser representado como una palabra de 8 bits. Se realizaron análisis estadísticos (véase Apéndice A), los cuales permitieron establecer que en la curva K-233, 64 bloques de memoria son suficientes para almacenar cualquier expansión NAF. Las memorias RAM son construidas a partir de bloques básicos BRAM integrados en las plataformas FPGA de Xilinx.

Por otro lado, la arquitectura presentada fue optimizada para una ventana de precómputo de $\omega = 8$ bits. Por lo tanto, una memoria ROM de 233x64 bits por coordenada es suficiente para almacenar los 64 múltiplos precalculados. Dado que los múltiplos de P son almacenados en coordenadas afines, bastan con dos memorias ROM para almacenar los puntos. Los múltiplos almacenados corresponden a múltiplos del punto generador de la curva $K-233$, como se requiere en el algoritmo generador de firma digital.

4.1. Unidad Principal de Proceso: Expansión $\omega\tau$NAF

La unidad de proceso dedicada a la generación de la expansión $\omega\tau$NAF está compuesta por: una unidad aritmética y lógica (UAL) que engloba las operaciones en aritmética entera así como por un bloque de registros. Una máquina de estados la cual contiene la microprogramación de los algoritmos correspondientes en palabras de un bus de control de 18 bits y la lógica de control para contar secuencias de ceros en la expansión generada.

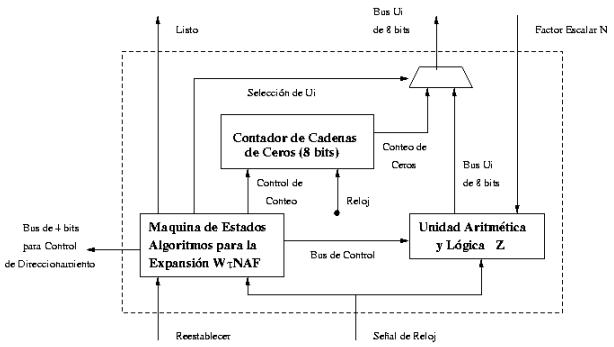

Figura 2: Unidad Principal de Proceso: Expansión τ NAF

El contador de ceros permite acumular la cantidad de ceros cada vez que empieza una secuencia consecutiva de ellos. La máquina de estados controla a través del bus *Selección Ui* si el valor que recién se computó en la expansión es distinto de cero o si se debe iniciar el conteo de los ceros que se ha acumulado. La máquina de estados rige la secuencia presente en el *Bus de Control*, el cuál provee a la unidad aritmética y lógica de los algoritmos para la generación de la expansión.

Unidad Aritmética y Lógica

La unidad aritmética y lógica (UAL) para los algoritmos de expansión fue diseñada para ejecutar el conjunto de micro operaciones atómicas previamente descrito. Dichas micro operaciones son ejecutadas dentro de un esquema de *pipe-line* de dos etapas: *Selección de Operadores-Cómputo Aritmético* y *Actualización de Registros*.

Las micro-operaciones realizadas por la unidad aritmética y lógica son establecidas por un bus de control de 19 bits. La unidad aritmética lógica es presentada en la Figura 3. Los operadores principales son la multiplicación de 128 bits y la adición-substracción de 256 bits. Dentro de los bloques dedicados a la selección de sus operandos, se encuentran operaciones simples basadas en corrimientos y reordenamiento de bits.

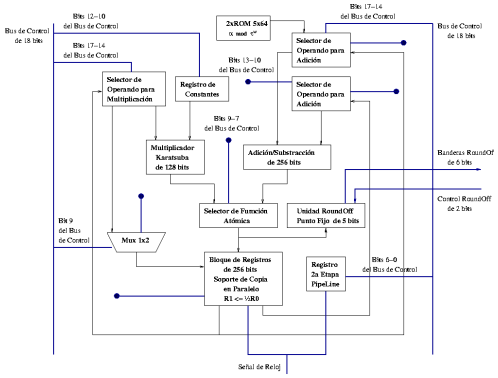

Figura 3: Unidad Aritmética y Lógica: Expansión τ NAF

Las comparaciones realizadas en los algoritmos son calculadas en el mismo ciclo correspondiente a la aritmética utilizando comparadores dedicados dentro de la unidad de registros. De esta manera es posible omitir un ciclo de reloj independiente para cada comparación.

4.2. Unidad Principal de Proceso: Curvas Elípticas

La aritmética base para esta unidad de control es la aritmética de campos finitos sobre la cual se define una curva elíptica. Para nuestro caso de estudio, se implementa la multiplicación escalar de un punto sobre la curva $K-233$. Dicha curva y su aritmética se define sobre el campo finito binario $GF(2^{233})$.

Los algoritmos que son implementados sobre esta arquitectura son tres. La ley de grupo sobre puntos de curvas elípticas implica dos algoritmos principales, la

adición entre puntos y el doblado de un punto correspondientes a los Algoritmos 2 y 3. El tercer algoritmo a implementar es la interpretación de la expansión $\omega\tau$NAF como una expresión polinomial de Horner, propuesto en versión paralela en el Algoritmo 4.

Otra característica importante incluida en esta arquitectura es la capacidad de aprovechar la secuencias de ceros consecutivos en una expansión $\omega\tau$NAF. Analizando el Algoritmo 4 se podrá corroborar que cada cero en la expansión corresponde con elevar al cuadrado cada coordenada de un punto elíptico. Una secuencia de $\omega-1$ ceros consecutivos implica elevar al cuadrado $\omega-1$ veces consecutivas un punto. La arquitectura presentada, aprovecha un buen diseño de la operación de elevación al cuadrado en campos finitos binarios para calcular eficientemente 7 cuadrados en un mismo ciclo de reloj, suficiente para una ventana de precómputo de 8 bits.

La arquitectura se divide en una unidad aritmética y lógica de campos finitos y una máquina de estados coordinando la aritmética para implementar los algoritmos propuestos.

Unidad Aritmética y Lógica

En la Figura 4 se presenta una arquitectura aritmética capaz de calcular una amplia gama de funciones aritméticas en un solo ciclo de reloj. La arquitectura está planteada para trabajar en el mismo esquema de pipeline de dos etapas propuesto para la unidad de proceso principal dedicada a la expansión.

La unidad aritmética principal tiene un multiplicador de campo finito Karatsuba-Offman. Dicho multiplicador demanda cerca del 70 % de los recursos de hardware de la arquitectura. También se presentan sumadores de campo bastantes económicos en requerimientos de tiempo y espacio.

La unidad aritmética es controlada por un bus de control de 32 bits capaz de generar una amplia variedad de combinaciones de operaciones aritméticas con hasta 4 operandos como entrada. La unidad está adaptada para trabajar de manera conjunta con el bloque de direccionamiento indirecto y leer los múltiplos de P almacenados en su memoria ROM como posibles operandos.

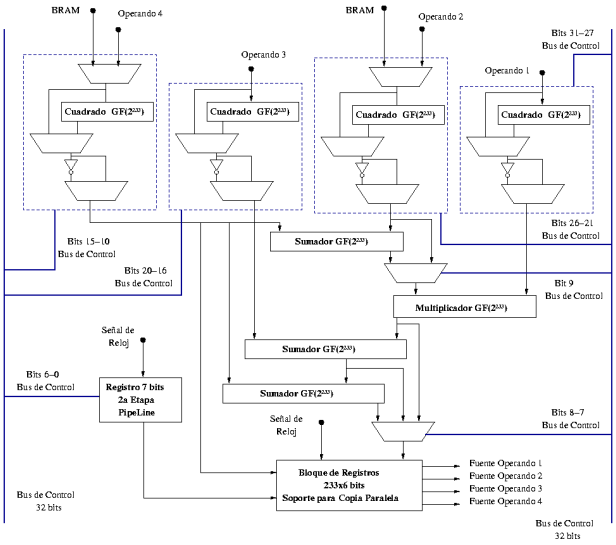

Figura 4: Unidad Aritmética y Lógica: Curvas Elípticas

5. Implementacion de Aritmética de Campos Finitos

La eficiencia de las arquitecturas propuestas en las Secciones anteriores depende de los algoritmos con los cuáles son programadas y, de manera aún más cercana, con la implementación de su aritmética básica. Los diseños presentados han sido optimizados para trabajar con la ley de grupo definida en el grupo de puntos sobre la curva elíptica $K-233$. Dicha curva ha sido definida sobre el campo finito binario $GF(2^{233})$ con el polinomio irreducible $z^{233}+z^{74}+1$.

Las operaciones de campo finito que conforman los algoritmos para definir la multiplicación escalar sobre grupos de curvas elípticas son la adición, multiplicación, cuadrado de campo y reducción de campo.

En el resto de esta Sección se presenta el diseño de las operaciones de campo binario utilizadas en la implementación de la arquitectura propuesta.

5.1. Multiplicación $GF(2^{233})$

La multiplicación de A y $B \in \mathrm{GF}[2^{233}]$ puede ser realizada en dos etapas. Primero se realiza un mul-

tiplicación polinomial de A y B para finalmente realizar una operación de reducción de campo módulo el polinomio irreducible $f(z) = z^{233}+x^{74}+1$. La multiplicación polinomial puede ser diseñada a partir del algoritmo *divide y vencerás* Karatsuba-Offman

El algoritmo de Karatsuba-Offman requiere de $O(n^{log_2 3})$ operaciones de bit para multiplicar dos enteros de n bits [6]. Sean $A = x^{\frac{m}{2}}A_H + A_L$ y $B = x^{\frac{m}{2}}B_H + B_L$, entonces:

$$\begin{aligned} C &= x^m A_H B_H + \\ &(A_H B_H + A_L B_L + (A_H + A_L)(B_H B_L))x^{\frac{m}{2}} \\ &+A_L B_L = x^m C_H + C_L \end{aligned} \quad (4)$$

De esta manera, la multiplicación Karatsuba-Offman define una multiplicación de m bits a partir de dos operandos polinomiales de $\frac{m}{2}$ bits. Una propiedad muy útil e interesante de un multiplicador Karatsuba-Offman es la posibilidad de ser paralelizado de manera eficiente en plataformas de hardware. En la ecuación 4 se encuentran tres multiplicaciones básicas $A_H B_H$, $A_L B_L$ y $(A_H + A_L)(B_H + B_L)$ independientes entre sí.

Este método puede ser aplicado de una manera muy simple en multiplicadores de 2^n bits. Sin embargo nuestra arquitectura necesita multiplicadores de 233 bits. Una solución es construir un multiplicador de 128 bits y completar el multiplicador a partir de una diseño Karatsuba-Offman a partir de multiplicadores de 105, 41 y 9 bits [6]. Como se muestra en el diagrama de la Figura 5.

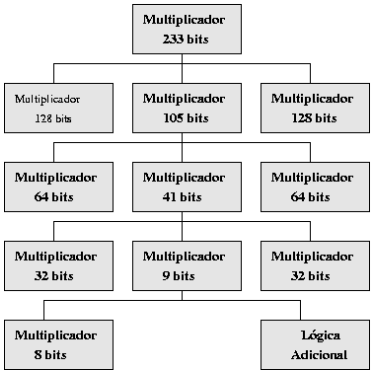

Figura 5: Árbol jerárquico de una estructura multiplicativa Karatsuba-Offman

Reducción Modular : Polinomio Irreducible $f(z) = z^{233} + z^{74} + 1$

La multiplicación polinomial es una operación básica en la arquitectura, sin embargo es necesario que el resultado de la multiplicación esté dentro del conjunto definido para el campo finito $GF(2^{233})$. El proceso de llevar la multiplicación C a un elemento del campo, C mod $z^{233}+z^{74}+1$, es llamado reducción modular.

En este trabajo, la reducción modular fue implementada a partir de sumas XOR y corrimientos. Debido a que el polinomio irreducible en el campo finito binario $GF(2^{233})$ es un trinomio, el proceso de reducción es particularmente eficiente. La Figura 6 describe el proceso de reducción de un polinomio de $2m-2$ bits, para $m = 233$, con el polinomio $z^{233} + z^{74} + 1$.

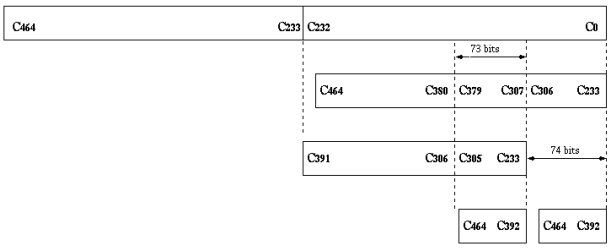

Figura 6: Arquitectura Reducción Modular: $z^{233}+z^{74}+1$

Cuadrado con Reducción Modular: $z^{233} + z^{74} + 1$

En campos finitos binarios, la operación de elevar al cuadrado un elemento de campo es una operación lineal. La representación binaria de $A(z)^2$ se obtiene insertando un bit cero entre bits consecutivos de una representación binaria de $A(z)$ seguido por una reducción modular.

Se observa que la mitad de la representación binaria de $A(z)^2$ está compuesta de ceros. Esta propiedad puede ser aprovechada durante el proceso de re-

Procedimiento 5 $A(z)^2 \bmod z^{233} + z^{74} + 1$

Entrada:

$A(z) = \sum_{i=0}^{m-1} a_i z^i$

Salida:

$A(z)^2 \bmod z^{233} + z^{74} + 1$

1. Para los bits i desde 0 hasta 73
   1.1 Si i es impar entonces $c_i = a_{\frac{i}{2}}$ xor $a_{\frac{i+392}{2}}$
   1.2 Si i es par entonces $c_i = a_{\frac{i+232}{2}}$
2. Para los bits i desde 74 hasta 146
   2.1 Si i es par $c_i = a_{\frac{i}{2}}$ xor $a_{\frac{i+318}{2}}$
   2.2 Si i es impar $c_i = a_{\frac{i+233}{2}}$ xor $a_{\frac{i+159}{2}}$
3. Para los bits i desde 147 hasta 232
   3.1 Si i es par $c_i = a_{\frac{i}{2}}$
   3.2 Si i es impar $c_i = a_{\frac{i+233}{2}}$ xor $a_{\frac{i+159}{2}}$
4. Regresar $C(z) = \sum_{i=0}^{m-1} c_i z^i$.

ducción modular. El Algoritmo 5.1 propone un algoritmo eficiente para calcular $A(z)^2 \bmod z^{233} + z^{74} + 1$ basado en el proceso de reducción modular clásico aprovechando esta característica. El desempeño obtenido supera incluso a una reducción modular genérica.

6. Resultados

La arquitectura descrita en este trabajo fue codificada en VHDL y simulada en el dispositivo Virtex 2 Xilinx XC2V4000. La arquitectura fue optimizada para la curvas elíptica K-233 recomendada por NIST y su correspondiente aritmética de campo $GF[2^{233}]$, y polinomio irreducible $\alpha^{233} + \alpha^{74} + 1$. El tamaño de la ventana es de 8 bits, por lo que se pre-calcularon y almacenaron en memoria RAM 2^{8-2} puntos de curva elíptica. La Tabla 1 muestra los requerimientos en hardware y desempeño en tiempo de los principales módulos en la arquitectura.

El número de ciclos de reloj necesarios para calcular la expansión $\omega\tau$NAF expansion con una ventana ω en un campo finito binario $GF[2^m]$ es: $\frac{m(4+w)}{w+1} + 17$

Por otro lado, el número de ciclos de reloj necesarios para calcular un multiplicación escalar con una ventana de pre-cómputo en un campo finito $GF[2^m]$ y un módulo de 7 cuadrados de campo es: $\frac{m(w+56)}{7w+7}$

Considerando que la unidad aritmética-lógica de

Módulo	**LUTs**	BRAM	Mult	**Reloj**
cuadrado de campo	153	—	—	—
7 cuadrados de campo	2003	—	—	—
Mult. Karatsuba-Offman	16674	—	—	—
Multiplicador $\mathbb{Z}$	0	—	48	—
ALU curvas Elípticas	29078	—	—	19.3ηS
ALU expansión	9541	2	48	37.1ηS
Diseño total	39762	11	48	36.2ηS

Cuadro 1: Recursos utilizados por los módulos principales de la arquitectura.

curvas elípticas alcanza un período de 19.3ηs y la unidad aritmética-lógica de la expansión $\omega\tau$NAF-EA tiene un período de 36.2ηS, se estima que contando con un reloj de división de frecuencia es posible obtener una multiplicación escalar en 16.61 μs. En la Tabla 2 se muestra una comparación de implementaciones FPGAs de multiplicación escalar de curvas elípticas definidas en $GF[2^m]$.

Autor	Campo	kP	LUTs	$\frac{1}{T \cdot LUT}$
[7]	$G2^{167}$)	210μs	3002	1.587
[8]	$GF(2^{191})$	75μs	10017	1.331
[9]	$GF(2^{163})$	76μs	10000	1.315
Aquí	$GF(2^{233})$	17,64μs	39762	1.425

Cuadro 2: Tabla Comparativa de Desempeños

7. Conclusiones

Se presentó el diseño de una arquitectura para calcular la operación de multiplicación escalar en curvas elípticas de Koblitz. Nuestro diseño alcanza una muy alta velocidad al precio de altos requerimientos en hardware. Considerando la figura de mérito velocidad/área, nuestro diseño obtiene un valor razonable dado por: $1/(Vel \cdot area) = [17{,}65\mu \cdot 39762]^{-1} = 1{,}425$. Nuestro diseño utilizó además un total de 48 multiplicadores de $18 \cdot 18$ bits y 11 bloques de la memoria BRAM disponibles en el dispositivo FPGA empleado. El proceso de verificación de diseño se realizó utilizando programas en Maple para la generación de vectores de prueba. En total se verificó el correcto funcionamiento del diseño para mil coeficientes escogidos al azar. Nuestro trabajo futuro incluye el uso de

multiplicadores seriales que permitan aliviar el costo actual en recursos de hardware.

Referencias

[1] Jerome A. Solinas. Efficient arithmetic on Koblitz curves. *Des. Codes Cryptography*, 19(2-3):195–249, 2000.

[2] Darrel Hankerson, Julio López Hernandez, and Alfred Menezes. Software implementation of elliptic curve cryptography over binary fields. In *CHES'00*, volume 1965 of *Lecture Notes in Computer Science*, pages 1–24, 2000.

[3] Darrel Hankerson, Alfred J. Menezes, and Scott Vanstone. *Guide to Elliptic Curve Cryptography*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2003.

[4] Neal Koblitz. CM-curves with good cryptographic properties. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 279–287. Springer, 1991.

[5] Jerome A. Solinas. An improved algorithm for arithmetic on a family of elliptic curves. In Burton S. Kaliski Jr., editor, *CRYPTO*, volume 1294 of *Lecture Notes in Computer Science*, pages 357–371. Springer, 1997.

[6] C. K. Koc F. Rodríguez-Henríquez. On fully parallel Karatsuba multipliers for $GF(2^m)$. In *Proceedings of International Conference on Computer Science and Technology CST*, pages 405–410, México, 2003. Acta Press.

[7] Gerardo Orlando and Christof Paar. A high performance reconfigurable elliptic curve processor for $GF(2^m)$. In *CHES '00*, volume 1965 of *Lecture Notes in Computer Science*, pages 41–56, London, UK, 2000. Springer-Verlag.

[8] J. Lutz. *High Performance Elliptic Curve cryptographic co-processor*. Master Thesis, University of Waterloo, 2004.

[9] Nazar A. Saqib, Francisco Rodríguez-Henríquez, and Arturo Díaz-Pérez. A parallel architecture for fast computation of elliptic curve scalar multiplication over $GF(2^m)$. In *RAW 2004*, pages 144–145, SantaFe, New Mexico, 2004. IEEE Computer Society Press.

Apéndice A: Longitud de la Expansión WτNAF

Solinas presenta en [1] un algoritmo para la generación de una expansión τNAF utilizando una ventana de precómputo de ancho ω. Solinas demuestra además que la longitud de una expansión τNAF, ℓ_{NAF}, para una curva elíptica $E_a[GF(2^m)]$ y de un equivalente modular reducido ρ de k, es siempre menor a $m + a$ donde a es el parámetro de la curva elíptica $E_a[GF(2^m)]$.

$$\ell_{NAF} \leq m + a + 3 \tag{5}$$

Se realizaron pruebas estadísticas de la longitud de la expansión WτNAF generadas por el algoritmo 1 para una ventana de precómputo $w = 8$, utilizando la curva elíptica $K - 233$, $E_0[GF(2^{233})]$.

La gráfica mostrada en la Figura 7 presenta las diferentes longitudes ℓ_{NAF} obtenidas para 1000 enteros positivos tomados al azar con expansiones generadas por el Algoritmo 1. Observe que tal algoritmo no presenta expansiones mayores a 235, según lo indicado por la Ecuación 5.

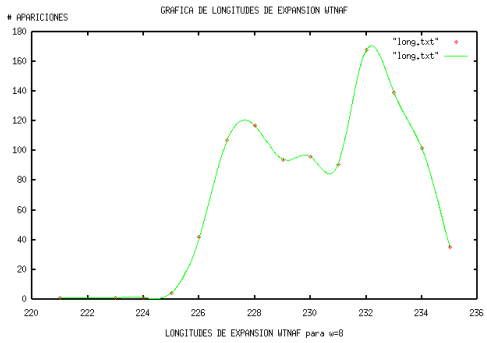

Figura 7: Longitudes de expansión $\omega\tau$NAF