Diseño de Unidades Aritmético Lógicas sobre cuerpos $GF(2^m)$ para aplicaciones Criptográficas

Luis Parrilla[1], Encarnación Castillo[1], Antonio García[1], Joaquín Olivares[2] y José M. Palomares[2]

Resumen— **En este artículo se presentan dos diseños de unidades aritmético lógicas sobre cuerpos finitos binarios $GF(2^m)$ para criptografía de curvas elípticas. La primera de ellas está orientada a sistemas de servicios criptográficos de altas prestaciones, permitiendo completar las operaciones en un número mínimo de ciclos de reloj, mejorando las implementaciones disponibles hasta la fecha. La segunda permite realizar cálculos criptográficos en dispositivos programables de bajo coste. Los diseños se han particularizado al cuerpo $GF(2^{233})$, y se han obtenido resultados para dispositivos Spartan 3, Spartan 6 y Virtex 6.**

Palabras clave—**criptografía, cuerpos finitos, unidad aritmético lógica, FPGAs.**

I. INTRODUCCIÓN

EL aumento constante de las comunicaciones y transacciones electrónicas ha generalizado el uso de herramientas criptográficas para mantener la seguridad y autenticidad de las mismas. Como consecuencia, cada vez un mayor porcentaje de los recursos de cálculo de las entidades que proporcionan algún tipo de servicio electrónico seguro se ha de dedicar a tareas relacionadas con la encriptación y la firma digital. En este sentido, los criptosistemas basados en curvas elípticas requieren de llaves de menor longitud que los sistemas clásicos como el RSA para mantener un mismo nivel de seguridad [1], lo que permite un ahorro en recursos y también una mayor aplicabilidad en sistemas como redes de sensores que disponen de anchos de banda reducidos. El incorporar coprocesadores criptográficos permite descargar a los servidores y/o clientes de toda esa carga de cálculo, a la vez que proporcionan una mayor flexibilidad para realizar cambios en los sistemas criptográficos sin afectar al resto del sistema. Los estándares sobre criptografía de curvas elípticas, como el IEEE 1363-2000 [1], recomiendan utilizar cuerpos finitos del tipo $GF(p)$ con p un número primo, $GF(2^m)$, o cuerpos de extensión óptima (Optimal Extension Fields, OEF), IEEE 1363a-2004 [2]. De ellos, en principio los que presentan mejores características para una implementación hardware son los cuerpos binarios $GF(2^m)$, y en este artículo se proponen dos arquitecturas para el diseño de una unidad aritmético lógica módulo (Modulo Arithmetic Logic Unit, MALU), sobre este tipo de cuerpos, particularizadas a $GF(2^{233})$. Una de las arquitecturas está pensada para grandes servidores, permitiendo la inversión en un número muy reducido de ciclos de reloj, presentando las mejores prestaciones hasta la fecha, pero con bastantes requerimientos en área. La otra propuesta es una MALU con mucha menor área, que necesita de mayores tiempos de cálculo, pero que puede implementarse en dispositivos programables de muy bajo coste como un dispositivo Spartan 3 de Xilinx.

[1] Departamento de Electrónica y Tecnología de Computadores. Universidad de Granada.
Email: lparrilla@ditec.ugr.es
[2] Departamento de Arquitectura y Tecnología de Computadores Universidad de Córdoba.

II. CURVAS ELÍPTICAS SOBRE CUERPOS FINITOS

Una curva elíptica E definida sobre un cuerpo finito $GF(q)$ consiste en un conjunto de puntos $P=(x_p,y_p)$ donde x_p e y_p son elementos de $GF(q)$ satisfaciendo la ecuación de Weirstrass [3], junto con el punto del infinito, representado por O. En el estándar [2], las curvas quedan definidas por dos coeficientes $a \in GF(q)$ y $b \in GF(q)$, denominados coeficientes de E. Si q es una potencia de 2, b ha de ser distinto de 0 en $GF(2^m)$, y los puntos $P=(x_p,y_p)$ sobre E (salvo el punto O) satisfacen la ecuación:

$$y_p^{\,3} + x_p y_p = x_p^{\,3} + a x_p^{\,2} + b \qquad (1)$$

Si $q=2^m$, los elementos del cuerpo pueden representarse utilizando bases polinómicas, a partir de un polinomio irreducible $f(t)$. En ese caso, un elemento a del cuerpo se representa mediante la cadena de bits:

$$a = (a_{m-1}..a_2 a_1 a_0) \qquad (2)$$

que corresponde al polinomio:

$$a(t) = a_{m-1}t^{m-1} + ..a_2 t^2 + a_1 t + a_0 \qquad (3)$$

donde los elementos a_i son elementos de $GF(2)$.

En el caso de cuerpos binarios, se definen el inverso del punto $P=(x,y)$ perteneciente a la curva E como:

$$-P=(x,x+y) \qquad (4)$$

Una de las características que permiten la utilización de las curvas elípticas para criptografía es la posibilidad de definir una operación interna dentro de la curva, denominada suma elíptica. Geométricamente, la suma de dos puntos P, Q, $R= P+Q$ viene dada por el punto R con la propiedad de que P, Q y $-R$ estén sobre la misma línea recta.

A partir de la suma puede definirse, dado un punto P de la curva E y un natural n, el producto escalar $n\cdot$P como:

$$P+\cdot^{\,n}\cdot \cdot +P \qquad (5)$$

Esta operación, que es el equivalente aditivo de la exponenciación en grupos abelianos multiplicativos, es la base de la criptografía con curvas elípticas.

III. OPERACIONES SOBRE CURVAS ELÍPTICAS

En el estándar [2] se establece el algoritmo mostrado en la Fig. 1 para realizar la suma elíptica en curvas definidas sobre cuerpos binarios utilizando bases polinómicas. Analizando el algoritmo se llega a la conclusión de que son necesarias 2 multiplicaciones, una elevación al cuadrado, y 1 inversión sobre el cuerpo, aparte de diversas sumas. Por otra parte, el algoritmo para el producto escalar se encuentra descrito en la Fig. 2, implicando la realización de sumas elípticas.
Como consecuencia, si se desea realizar un coprocesador criptográfico que realice el producto escalar es necesario disponer de una unidad aritmético lógica módulo que realice las siguientes operaciones básicas sobre el cuerpo:

- Suma
- Elevación al cuadrado
- Multiplicación
- Inversión

Nótese que todas estas operaciones se han de realizar entre polinomios, resultando la multiplicación y el cálculo del inverso multiplicativo las operaciones más costosas en términos de área y retardo.

Entrada: Dos puntos $P_0(x_0,y_0)$ y $P_1(x_1,y_2)$ de la curva
Salida: El punto $P_{2:}=P_0+P_1$

1. Si $P0 = O$, entonces $P2 \leftarrow P1$. Fin
2. If $P1 = O$, entonces $P2 \leftarrow P0$. Fin.
3. Si $x_0 \neq x_1$ entonces
   3.1 Hacer $\lambda \leftarrow (y_0 + y_1)/(x_0 + x_1)$.
   3.2 Hacer $x_2 \leftarrow a + \lambda^2 + \lambda + x_0 + x_1$.
   3.3 Ir al paso 7.
4. Si $y_0 \neq y_1$, entonces $P_2 \leftarrow O$. Fin.
5. Si $x_1 = 0$, entonces $P2 \leftarrow O$. Fin.
6. Hacer
   6.1 $\lambda \leftarrow x_1 + y_1/x_1$.
   6.2 $x_2 \leftarrow a + \lambda_2 + \lambda$.
7. $y_2 \leftarrow (x_1 + x_2)\,\lambda + x_2 + y_1$.
8. $P_2 \leftarrow (x_2, y_2)$.
9. Fin

Fig. 1. Algoritmo 1: Suma elíptica en curvas definidas sobre cuerpos $GF(2^m)$ descritos en base polinómica.

A. Suma y multiplicación sobre cuerpos GF(2^m)

Tal y como se ha comentado en la Sección II, la suma sobre cuerpos finitos binarios se efectúa sumando bit a bit la representación correspondiente, sin mayores dificultades. La multiplicación precisa sin embargo de la elección de una base, que en caso de ser de tipo polinómico establece una representación en términos del denominado polinomio de definición del cuerpo $f(t)$, de grado m. En ese caso, los m bits de la representación corresponden a los coeficientes de un polinomio de grado m-1, y la multiplicación se realiza multiplicando los polinomios correspondientes y reduciendo modularmente sobre $f(t)$, obteniéndose otro polinomio de grado m-1. Nótese que $f(t)$ debe ser un polinomio irreducible. En este artículo, los ejemplos de implementación se van a realizar sobre el cuerpo $GF(2^{233})/(t^{233} + t^{74} + 1)$ (es decir, el polinomio del cuerpo será $f(t)= t^{233} + t^{74} + 1$), utilizado en los estándares de la IEEE y del NIST [4]).

Entrada: Un entero n y un punto P de la curva
Salida: El punto nP

1. Si $n = 0$, entonces devolver `O. Fin.`
2. Si $n < 0$, entonces hacer $Q \leftarrow (-P)$ y $k \leftarrow (-n)$; else set $Q \leftarrow P$ and $k \leftarrow n$.
3. Sea $h_l\, h_{l-1} \ldots h_1\, h_0$ la representación binaria de $3k$, donde el MSB h_l es 1.
4. Sea $k_l\, k_{l-1} \ldots k_1\, k_0$ la representación binaria de k.
5. Hacer $S \leftarrow Q$.
6. Para i desde $l - 1$ bajando hasta 1 hacer
   Hacer $S \leftarrow 2S$.
   Si $h_i = 1$ y $k_i = 0$, entonces
   calcular $S \leftarrow S + Q$
   Si $h_i = 0$ y $k_i = 1$, entonces
   calcular $S \leftarrow S - Q$
7. Devolver S.
8. Terminar

Fig. 2. Algoritmo 2: Producto escalar en curvas elípticas.

Por tanto, la realización de multiplicaciones dentro del cuerpo requiere de dos operaciones:

- Producto de polinomios de grado m
- Reducción modular sobre $f(t)$

El producto se puede realizar utilizando circuitos combinacionales o secuenciales, en función de los requerimientos de área y/o retardo. En la Sección V se estudian y seleccionan diversas posibilidades de implementación de multiplicadores para utilizar en las MALUs propuestas. En cuanto a la reducción modular se puede utilizar la implementación directa de matrices de reducción modular, lo que se traduce en una red de puertas XOR, o bien alguna estructura secuencial para realizar la división por el polinomio de definición $f(t)$. En todo caso, las soluciones secuenciales no presentan ventajas claras en área ni en retardo y se utilizan siempre matrices de reducción modular.

B. Inversión en cuerpos GF(2^m)

La inversión es la operación más compleja dentro de cuerpos finitos binarios. Existen diversas soluciones, todas ellas de carácter secuencial, y basadas en dos herramientas matemáticas distintas:

- Algorimo de Euclides extendido (EEA). Existen implementaciones eficientes como la presentada en [5], que permiten realizar la inversión en m ciclos de reloj a frecuencias elevadas. No

obstante, para implementaciones de altas prestaciones este número de ciclos de reloj puede resultar elevado

- Pequeño teorema de Fermat (LTF). El pequeño teorema de Fermat establece que el inverso en un cuerpo finito puede obtenerse como:

$$p^{-1} = p^{2^m-2} = (p^{2^{m-1}-1})^2 \quad (6)$$

mediante una aplicación sucesiva de elevaciones al cuadrado como la descrita en el estándar [1], se puede completar la inversión en *m* ciclos de reloj. Otra posibilidad es utilizar el algoritmo de Itoh-Tsujii (ITA) [6] que minimiza el número de pasos en el cálculo de la exponenciación utilizando cadenas aditivas mínimas o de Brauer. En [7] y [8] se presentan implementaciones que permiten la inversión sobre $GF(2^{233})$ en unos 27-30 ciclos de reloj, mejorando sensiblemente las prestaciones del EEA.

En este trabajo se propone utilizar una implementación del algoritmo ITA que permite la inversión en sólo 10-14 ciclos de reloj sobre $GF(2^{233})$ o $GF(2^{409})$.

C. Elevación al cuadrado

La elevación al cuadrado es necesaria tanto en el algoritmo de la Fig. 1 como en la inversión si se realiza a partir del LTF. Una posibilidad es utilizar el multiplicador, y multiplicar el polinomio por sí mismo, pero resulta mucho más eficiente aprovechar que se está trabajando sobre en un cuerpo finito, y por tanto el cuadrado puede obtenerse como:

$$p^2 = C \cdot p \quad (7)$$

donde *C* es una matrix *m*x*m*. La implementación se puede llevar a cabo mediante una red de puertas XOR de sólo 153 puertas para $GF(2^{233})$, con un retardo muy reducido.

IV. DESCRIPCIÓN GENERAL DE LA MALU

La unidad MALU propuesta realizará cuatro operaciones fundamentales, la suma, elevación al cuadrado, multiplicación e inversión. Se utilizarán dos entradas de control para seleccionar la operación a realizar. Además se tendrán dos entradas de *m* bits para los polinomios con los que se va a operar, una salida de *m* bits con el resultado de la operación, una entrada de señalización, *inicio*, que deberá activarse durante un pulso de reloj para que la MALU inicie la operación, una salida de señalización *fin*, con la que la MALU indicará durante un pulso de *reloj* que ha terminado la operación, y las señales de *reloj* y *reset*. En la Tabla I se presentan los distintos puertos de la MALU, resumiendo los distintos modos de operación.

El diseño se podría abordar realizando por separado un módulo que realice cada una de las cuatro operaciones, y después definir los buses y conexiones que fueran necesarios, así como la unidad de control. Sin embargo, el hecho de que la inversión precisa de operaciones de multiplicación aconseja reutilizar el multiplicador y realizar un diseño conjunto de los distintos elementos.

La Fig. 3 muestra el diseño propuesto, en el que se tiene un sumador cuya salida se puede registrar en el registro R, un multiplicador cuyo resultado puede almacenarse también en R, y una matriz de elevación al cuadrado que proporciona a la salida el cuadrado de R. Para realizar la inversión se dispone de un módulo de exponenciación, que se particularizará en función del algoritmo utilizado, y de un registro RC para acumular exponenciaciones. El diseño se completa con los multiplexores necesarios y la unidad de control que gestionará los distintos elementos para completar las operaciones deseadas.

TABLA I

PUERTOS Y MODOS DE OPERACIÓN DE LA MALU PROPUESTA

Puerto	Tipo	Descripción
p	Entrada, m bits	Uno de los polinomios de entrada
q	Entrada, m bits	Otro poliniomio de entrada
salida	Salida, m bits	Resultado de la operación
oper	Entrada, 2 bits	Entrada que selecciona la operación a realizar: 00 -> suma 01 -> cuadrado 10 -> multiplicación 11 -> inversión
inicio	Entrada, 1 bit	Un pulso en esta entrada indica a la MALU que comience la operación seleccionada.
fin	Salida, 1 bit	Un pulso en esta salida indica que la MALU ha terminado la operación seleccionada.
reloj	Entrada, 1 bit	Entrada de reloj general del sistema
reset	Entrada, 1 bit	Entrada de reset general del sistema

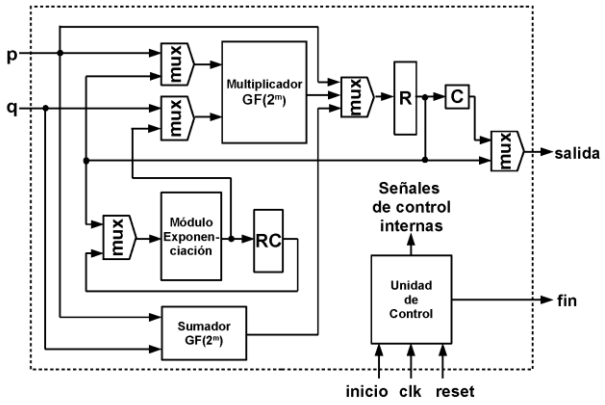

Fig. 3. Diagrama de bloques general para el diseño de MALU propuesto.

Partiendo de este esquema general, se van a realizar dos diseños de los módulos particularizados a $GF(2^{233})$, uno de los cuerpos descritos en los estándares sobre curvas elípticas. Uno de los diseños, que denominaremos MALU2MHP estará orientado a conseguir altas prestaciones, y se implementará sobre dispositivos Virtex 6 de Xilinx. El otro, denominado MALU2MLC tratará de optimizar todo lo posible el área, procurando obtener un diseño de bajo coste que se probará en dispositivos Spartan 3A y Spartan 6.

V. IMPLEMENTACIÓN DE LOS MÓDULOS INTEGRANTES DE LA MALU

A continuación se describen las implementaciones de los distintos bloques integrantes de la MALU, en función del tipo de diseño (altas prestaciones o bajo coste) que se desee obtener, particularizados a $GF(2^{233})$

A. Diseño del sumador sobre GF(2^233)

La suma se realiza directamente utilizando una red de 233 puertas XOR. El área y el retardo son mínimos, por lo que se utilizará este mismo diseño para las dos implementaciones, MALU2MHP y MALU2MLC.

B. Diseño del multiplicador sobre GF(2^{233})

Para realizar el producto de polinomios existen diversas soluciones, tanto de tipo combinacional como secuencial. Entre las soluciones combinacionales destaca el multiplicador de Karatsuba-Offman (KOA) [9], de tipo recursivo, y cuyo número de puertas aumenta como $m^{\log_2(3),}$ en lugar de m^2 como ocurre con el multiplicador clásico, a costa de un mayor retardo. En [10] se presenta una mejora del multiplicador de Karatsuba, que evita solapamientos y que denominaremos NOKOA (Non-overlapping KOA). La Tabla II muestra la comparación de las complejidades y retardos de estos tres multiplicadores, donde *TA* es el retardo correspondiente a una puerta AND, y *TX* el correspondiente a una puerta XOR.

TABLA II

NÚMERO DE PUERTAS AND, XOR Y RETARDO PARA DISTINTOS MULTIPLICADORES DE POLINOMIOS COMBINACIONALES

Algoritmo	#AND	#XOR	Retardo
Clásico	m^2	$(m\text{-}1)^2$	$\log_2(m)TX+TA$
KOA	$m^{\log_2(3)}$	$6\,m^{\log_2(3)}\text{-}8m+2$	$(3\log_2(m)\text{-}1)\ TX+TA$
NOKOA	$m^{\log_2(3)}$	$6\,m^{\log_2(3)}\text{-}8m+2$	$(2\log_2(m))\ TX+TA$

Debido al menor retardo del multiplicador clásico, y aprovechando la recursividad de los multiplicadores KOA y NOKOA, se suelen utilizar multiplicadores híbridos, en los que se rompe la recursividad a partir de un determinado valor de m, habitualmente entre 8 y 32, introduciendo un multiplicador clásico. En este trabajo se ha decidido utilizar multiplicadores NOKOA híbridos cortando la recursividad en m=16, para la implementación de altas prestaciones MALU2MHP, buscando también una cierta contención en el área.

En cuanto a la implementación de bajo coste, se han considerado dos posibilidades:

- Aprovechar la recursividad del multiplicador NOKOA para segmentar y reutilizar los multiplicadores del nivel siguiente. Si se hace esto en un nivel, se consigue reducir el área a la mitad a costa de necesitar 3 ciclos de reloj para completar la multiplicación. Si se hace en dos niveles, se necesitan 9 ciclos de reloj, y el área de reduce a la quinta parte aproximadamente.
- Utilizar un multiplicador secuencial de acumulación y desplazamiento, con un área muy reducida, pero precisando de m ciclos de rejoj para completar la multiplicación.

En la Tabla III se muestran los resultados de síntesis para los multiplicados NOKOA, NOKOA3C (reutilización en un nivel, 3 ciclos de reloj), NOKOA9C (reutilización en dos niveles, 9 ciclos de reloj), y MAD (multiplicador de acumulación y desplazamiento), para dispositivos Spartan 3AN (XCS700AN-4), Spartan 6 (Xc6SLX45T) y Virtex 6 (XC6VLX240T), sobre $GF(2^{233})$.

Para la implementación MALU2MHP la mejor opción (si se dispone de área suficiente) es utilizar NOKOA o en todo caso NOKOA3C si se necesita reducir el área y/o el retardo (a costa de triplicar el número de ciclos de reloj). Para la implementación de bajo coste MALU2MLC, en dispositivos Spartan 6 el ahorro en área que presenta el multiplicador MAD frente al NOKOA9C no justifica la gran diferencia en el número de ciclos. En dispositivos Spartan 3, sí puede tener sentido en caso de resultar imprescindible una reducción del área. En todo caso, se va a utilizar el multiplicador NOKOA9C para la MALU2MLC.

TABLA III

RESULTADOS DE SÍNTESIS PARA DISTINTOS TIPOS DE MULTIPLICADORES SOBRE GF(2^{233})

Disp.	Mul.	#LUTS	Ret. (ns)	#Ciclos	Tiempo total (ns)
Spartan 3	MAD	1422	5.0	233	1160
Spartan 3	NOKOA9c	3099	13.5	9	121.6
Spartan 3	NOKOA3c	6385	14.1	3	42.3
Spartan 6	MAD	1464	3.5	233	818.8
Spartan 6	NOKOA9c	2333	7.9	9	71.4
Spartan 6	NOKOA3c	4865	8.0	3	23.9
Spartan 6	NOKOA	11366	15.9	1	15.9
Virtex 6	NOKOA9c	2315	4.9	9	43.7
Virtex 6	NOKOA3c	4856	5.0	3	15.0
Virtex 6	NOKOA	11366	7.4	1	7.4

C. Implementación del inversor

Tal y como se ha comentado en la sección anterior, el algoritmo ITA permite realizar la inversión en un número muy reducido de ciclos de reloj, por lo que se ha considerado para la implementación MALU2MHP. Si se define $A_k(p) = p^{2^k-1}$, se puede demostrar que [7]:

$$A_{k+j}(p) = (A_j)^{2^k} A_k = (A_k)^{2^j} A_j \qquad (8)$$

y por tanto, $p^{2^{m-1}-1} = A_{m-1}$ se puede expresar en términos de matrices de exponenciación aplicadas sucesivamente. La cuestión es seleccionar k y j en cada paso para ir de A_1 hasta A_{m-1} utilizando el menor número de pasos intermedios, construyendo una cadena aditiva en la que cada nuevo valor $(k+j)$ se puede escribir como la suma de dos valores previos. Este tipo de cadenas aditivas se denominan cadenas de Brauer [11]. A modo de ejemplo, si se considera el cuerpo $GF(2^{233})$, m-1 = 232, y se puede utilizar la siguiente cadena mínima de Brauer:

$$U1_{232} = \{1, 2, 3, 6, 7, 14, 28, 29, 58, 116, 232\} \qquad (9)$$

Nótese como cada elemento de la cadena se puede escribir como la suma de dos elementos anteriores

(2=1+1, 3=2+1, 6=3+3, etc. desde 1 hasta 232) resultando:

$$A_1 = p\,;\; A_2 = A_{1+1} = (A_1)^{2^1} A_1 = p^{2^2-1}\,;\; A_3 = A_{2+1} = (A_2)^{2^1} A_1 = p^{2^3-1}\,;$$
$$\ldots;\; A_{232} = A_{116+116} = (A_{116})^{2^{116}} A_{116} = p^{2^{232}-1} \qquad (10)$$

Cada paso implica realizar una exponenciación y una multiplicación. Si se utiliza la matriz *C* repetidamente para realizar la exponenciación, se puede completar la cadena de Brauer sin un impacto excesivo en el área (*C* requiere 153 puertas XOR). Para mejorar las prestaciones en [7, 8] se propone utilizar cadenas de matrices *C* con un número de elementos que iguale el retardo con el del multiplicador. Con una cadena de 12 matrices *C*, el último paso implicaría utilizar 10 ciclos de reloj (hay que elevar a 2^{116} en ese paso, pero en cada ciclo sólo se puede elevar a 2^{12}), consiguiendo realizar la inversión completa en 36 ciclos de reloj [7]. En este trabajo proponemos utilizar matrices de exponenciación completas (es decir, construir la matriz C^{116} que realiza directamente la elevación a 2^{116}, C^{58} para elevar a 2^{58}, etc), de manera que cada paso requiera sólo un ciclo de reloj. La implementación obtenida, que denominaremos *full-ITA* permite la inversión en 10 ciclos de reloj, aunque a costa de un área elevada. En la Fig. 4 se muestra el número de puertas XOR que precisa la matriz C^n para $GF(2^{233})$. El número de puertas crece rápidamente con *n* hasta estabilizarse alrededor de 26400 para *n*>=18. Por tanto, el inversor full-ITA debe usarse solamente para la implementación de altas prestaciones MALU2MHP.

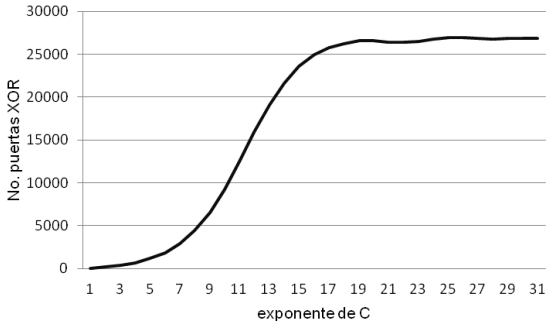

Fig. 4. Número de puertas en función del exponente de C.

Para reducir el área y el retardo, se ha realizado una optimización del módulo de exponenciación, eliminando C^{116} y C^{56} a costa de incrementar el número de ciclos de reloj a 14, obteniendo la arquitectura denominada *full_op2-ITA*. En la Tabla IV se muestran los resultados de implementación y retardo para los módulos de exponenciación.

Para la implementación MALU2MLC, se pueden aprovechar las ideas presentadas en [7] y [8] para ahorrar algunos ciclos de reloj. Así, si se utilizan matrices de exponenciación de la forma $4^k = 2^{2^k}$, resulta el algoritmo conocido como *quad-ITA* [7]. Ahora, la matriz de exponenciación básica es $Q=C^2$, y se tiene la siguiente expresión:

$$B_{k+j}(p) = (B_j)^{4^k} B_k = (B_k)^{4^j} B_j \qquad (11)$$

Aplicando (11) a la cadena aditiva $U1_{232}$, se precisan solo 9 pasos para completarla puesto que en el elemento 116 se tiene:

$$B_{116} = B_{58+58}(p) = (B_{58})^{4^{58}} B_{58} = p^{4^{116}-1} = p^{2^{232}-1} \qquad (12)$$

y el cálculo se puede completar en 30 ciclos de reloj si se utilizar una cadena de 8 matrices *Q* para la exponenciación. De manera análoga se puede definir el *octate-ITA* [8], y en general el algoritmo, 2^n-ITA, ahorrando pasos en la cadena aditiva correspondiente. En [8] se ha obtenido una implementación que necesita 27 ciclos de reloj para completar la inversión sobre $GF(2^{233})$ usando 2^4-ITA. Para la implementación MALU2MLC se ha decidido utilizar el algoritmo 2^4-ITA con una cadena de 4 matrices C^4, reduciendo notablemente el área. Al tener que repetir varias veces el proceso de exponenciación para completar cada paso, es necesario añadir un registro a la arquitectura de la Fig. 3, para la implementación MALU2MLC.

TABLA IV

RESULTADOS DE SÍNTESIS PARA DISTINTOS MÓDULOS DE EXPONENCIACIÓN SOBRE $GF(2^{233})$

Disp.	Mod. exponenciacion	#LUTS	Ret. (ns)
Spartan 3	2^4 ITA longitud 4	2534	17.1
Spartan 3	full_op2-ITA	9856	13.6
Spartan 3	full-ITA	21658 *	12.3
Spartan 6	2^4 ITA longitud 4	1883	10.0
Spartan 6	full_op2-ITA	5717	9.0
Spartan 6	full-ITA	13405	8.8
Virtex 6	2^4 ITA longitud 4	1888	6.7
Virtex 6	full_op2-ITA	5725	6.0
Virtex 6	full-ITA	13381	7.0

*Excede de la capacidad del dispositivo

VI. IMPLEMENTACIÓN DE LA MALU SOBRE $GF(2^{233})$

Teniendo en cuenta los diseños obtenidos para el multiplicador y el inversor, se han realizado dos tipos de implementaciones. Para sistemas de altas prestaciones se han probado dos variantes de la denominada MALU2MHP, una con el inversor full-ITA y otra con el full_op2-ITA. Asimismo para cada diseño se han obtenido resultados de síntesis y post "place&route" sobre un dispositivo de la familia Virtex 6 (XC6VLX240T), tal y como se muestra en la Tabla V. Los resultados de síntesis permiten realizar comparaciones con otras implementaciones que sólo proporcionan este tipo de resultados. Por ejemplo, los inversores presentados en [7] y [8] precisan de 30 ciclos de reloj y 27 para los diseños *quad-ITA* y *2^4-ITA*, respectivamente, con tiempos totales para completar la inversión de 233.7ns y 247.3ns. Si se atiende a los resultados de síntesis de la Tabla V, los resultados obtenidos por MALU2MHP son netamente mejores, con sólo 10 ciclos de reloj para completar la inversión y un total de 83.4ns.

Por otra parte, los resultados "place&route", muestran que los retardos reales son mayores y aumentan los tiempos para completar el resultado, aunque el área disminuye alrededor de un 17%. Asimismo, se comprueba que a pesar de los 4 ciclos de reloj adicionales que implica el utilizar el inversor *full_op2-ITA* frente al *full-ITA*, el primero presenta menor retardo obteniendo un menor tiempo total de cálculo, aparte del ahorro en área que se consigue. A la vista de estos resultados, se ha optado por utilizar finalmente *full_op2-ITA* para la implementación MALU2MHP.

TABLA V

RESULTADOS DE SÍNTESIS Y PLACE & ROUTE PARA LA IMPLEMENTACIÓN MALU2MHP SOBRE GF(2^{233})

Disp./ Resultado	Mul/ Inversor	#LUTS	Ret. (ns)	#Ciclos	Tiempo total (ns)
Virtex 6 síntesis	NOKOA full-ITA	27086	8.3	10	83.4
Virtex 6 P&R	NOKOA full-ITA	21498	33.8	10	338
Virtex 6 Síntesis	NOKOA full_op2-ITA	18533	15.3	14	214.5
Virtex 6 P&R	NOKOA full_op2-ITA	15796	23.4	14	327.3

En cuanto a la implementación de bajo coste, MALU2MLC, se ha optado por el multiplicador NOKOA9c y un inversor 2^4-ITA, obteniendo resultados para dispositivos Spartan 3 y Spartan 6 . La Tabla VI presenta los mismos, tanto para síntesis, como para "place&route". Como puede observarse, se tienen áreas muy contenidas, y la MALU ofrece los resultados en unos tiempos muy razonables, en 102 ciclos de reloj, un número inferior a los 233 que necesitan implementaciones basadas en inversores del tipo Algoritmo de Euclides Extendido.

TABLA VI

RESULTADOS DE SÍNTESIS Y PLACE & ROUTE PARA LA IMPLEMENTACIÓN MALU2MLC SOBRE GF(2^{233})

Disp./ Resultado	Mul/ Inversor	#LUTS	Ret. (ns)	#Ciclos	Tiempo total (ns)
Spartan 3 Sintesis	NOKOA9C 2^4-ITA	8999	26,3	102	2683,6
Spartan 3 P&R	NOKOA9C 2^4-ITA	7292	38,2	102	3891,3
Spartan 6 Síntesis	NOKOA9C 2^4-ITA	5687	24,5	102	2497
Spartan 6 P&R	NOKOA9C 2^4-ITA	4972	24,5	102	2499

VII. CONCLUSIONES

En este trabajo se han presentado el diseño de una Unidad Aritmético Lógica Módulo para trabajar sobre cuerpos finitos binarios, orientada a su utilización como núcleo de coprocesadores criptográficos sobre curvas elípticas. Se ha mostrado la viabilidad del diseño, y se han presentado dos implementaciones, una de ellas de altas prestaciones, orientada a servidores criptográficos y que se ha denominado MALU2MHP, y otra para clientes, buscando contener el área y que se ha denominado MALU2MLC. Las MALUs desarrolladas permiten realizar cuatro operaciones: suma, elevación al cuadrado, multiplicación e inversión sobre cuerpos $GF(2^m)$.

Los resultados obtenidos muestran que la implementación MALU2MHP puede obtener el resultado de la operación deseada entre 1 y 14 ciclos de reloj (1 para suma, elevación al cuadrado y multiplicación; 14 para inversión), con un tiempo total de 338ns sobre dispositivos Virtex 6. En cuanto a la implementación de bajo coste, MALU2LC precisa de 102 ciclos de reloj para completar la operación de inversión, pero precisamente mucha menor área. Los resultados muestran que se puede implementar incluso en un dispositivo Spartan 3AN.

REFERENCIAS

[1] IEEE, *"IEEE Standard Specifications for Public-Key Cryptography"*, IEEE Std 1363-2000, 2000.

[2] IEEE, *"IEEE Standard Specifications for Public-Key Cryptography- Amendment 1: Additional Techniques",* IEEE Std 1363a-2004 (Amendment to IEEE Std. 1363-2000) , 2004.

[3] N. Koblitz, "Elliptic curve cryptosystems". Mathematics of Computation, Vol. 48, No. 177, pp. 203–209. 1987.

[4] NIST, FIPS PUB 186-3*, "Digital Signature Standard"* , Federal Information Processing Standard, National Institute of Standards and Technology (NIST), Computer Security, Junio 2009.

[5] Z. Yan, D.V. Sarwate*, "New systolic architectures for inversion and division in GF(2^m)"*, IEEE Transactions on Computers, vol.52, no.11, pp. 1514- 1519, Nov. 2003

[6] T. Itoh, S. Tsujii, *"A fast algorithm for computing multiplicative inverses in GF(2^m) using normal bases"*, Inf. Comput., Vol. 78, No. 3, pp. 171–177, 1988.

[7] C. Rebeiro, S.S. Roy, D.S. Reddy, D. Mukhopadhyay, *"Revisiting the Itoh-Tsujii Inversion Algorithm for FPGA Platforms",* Very Large Scale Integration (VLSI) Systems, IEEE Transactions on , vol.19, no.8, pp.1508-1512, Aug. 2011.

[8] S.S. Roy, C. Rebeiro, D. Mukhopadhyay, *"Theoretical modeling of the Itoh-Tsujii Inversion algorithm for enhanced performance on k-LUT based FPGAs",* Design, Automation & Test in Europe Conference & Exhibition (DATE), 2011 , vol., no., pp.1-6, 14-18 March 2011.

[9] J. von zur Gathen, J. Shokrollahi, *"Efficient FPGA-based Karatsuba multipliers for polynomials over F2"*, B. Preneel and S. Tavares (Eds.):SAC 2005, LNCS, Vol. 3897, pp. 359–369, 2006.

[10] H. Fan, J. Sun, M. Gu, K.Y. Lam, *"Overlap-free Karatsuba-Offman Polynomial Multiplication Algorithms",* IET Information security, vol. 4, no. 1, pp. 8-14.

[11] A. Brauer, *"On addition chains"*. Bull. Amer. Math. Cos., Vol. 45, pp. 736-739. 1939