Co-procesador matemático de aritmética entera basado en un FPGA

Joel Noyola Bautista, Oscar Alvarado-Nava y Felipe Monroy Pérez*
Departamento de Electrónica,*Departamento de Ciencias Básicas
Universidad Autónoma Metropolitana, Unidad Azcapotzalco, México D.F., México
joelnb88@gmail.com, {oan,fmp}@correo.azc.uam.mx

***Abstract*—Los sistemas criptográficos modernos de llave pública requieren de operaciones especiales como la multiplicación y exponenciación modular de enteros muy largos, enteros representandos desde 512 a 2048 bits. Si dichas operaciones se llevan a cabo a través de un programa que se ejecuta en un sistema de cómputo tradicional, el tiempo de procesamiento es alto debido a que al utilizar enteros que son representados con más de 32 o 64 bits no pueden ser procesados directamente en las unidades funcionales de un CPU de propósito general, siendo necesario crear a través de software, arreglos de enteros de 32 bits para ser procesados en partes. Aplicaciones de sofware orientados al cálculo matemático, como MAPLE y MATHEMATICA, realizan las operaciones de multiplicación y exponenciación modular creando tipos de datos especiales y utilizando algoritmos eficientes para mejorar el tiempo de ejecución. El algoritmo de Montgomery para la multiplicación modular es considerado el algoritmo más rápido para calcular $a \cdot b \ mod \ n$ donde a, b y n son enteros muy largos.**

A través de dispositivos programables, como los FPGAs, es posible implementar circuitos digitales que lleven a cabo el cálculo de operaciones aritméticas de manera eficiente aprovechando el paralelismo inherente del hardware. El presente trabajo describe el diseño de un co-procesador matemático de aritmética entera para números representados con 1024 bits en un FPGA. El co-procesador realiza la multiplicación modular a través de la implementación en hardware del algoritmo de Montgomery acelerando así las operaciones de los sistemas criptográficos. El coprocesador fue agregado a un sistema de cómputo tradicional a través del sistema de buses.

***Index Terms*—Aritmética, Coprocesador, Criptografía, FPGA, Montgomery, Security.**

I. Introducción

La motivación para el estudio de algoritmos eficientes y de alta velocidad para la multiplicación modular viene de las aplicaciones en la criptografía, por ejemplo para la generación de llaves públicas [1]. Ciertamente una de los avances más útiles e interesantes ha sido la introducción al llamado algoritmo de multiplicación modular de Montgomery. El algoritmo de Montgomery [2] es usado para acelerar la multiplicación modular y la exponenciación modular, el algoritmo calcula

$$MonPro(a,b) = a \cdot b \cdot \ r^{-1} \ mod \ n \tag{1}$$

dados a, $b < n$ y r tales que $mod(n,r) = 1$. Por lo tanto el algoritmo trabaja con algún r, el cual es primo relativo con n. El algoritmo deberá realizar varias operaciones aritméticas las cuales se pueden llevar de manera sencilla y eficiente si se selecciona r como un número potencia de 2.

En general, el algoritmo de Montgomery es considerado el más rápido de los algortimos para calcular, $x \cdot y$ mod n, en computadoras cuando los valores de x, y y n son muy grandes [3]. En la siguiente sección se describe el algoritmo de Montgomery para la multiplicación modular para su implementación en software.

II. Algoritmo para la multiplicacion de Montgomery

Supongamos que se quiere calcular $x \cdot y$ mod n, elegimos un número entero postivo r más grande que n y primo realtivo a n. El valor de r deberá ser 2^k para algún entero positivo k, se elige con esa característica para aprovechar que la multiplicación, la división y el módulo por r se pueden hacer fácilmente con operaciones lógicas y corrimientos en los registros de una unidad de procesamiento.

Sea n el módulo, un entero de $k-bits$, es decir, $2^{k-1} \leq n < 2^k$, y sea r 2^k. El algoritmo de multiplicación de Montgomery requiere que r y n sean primos relativos, es decir, $mcd(r,n) = mcd(2^k,n) = 1$. Este requisito se satisface si n es impar. Con la finalidad de describir el algoritmo de multiplicación de Montgomery, primero definimos el residuo-n de un entero $a < n$ como $\bar{a} = a \cdot r \ (mod \ n)$. Con esto es fácil mostrar que el conjunto

$$\{a \cdot r \ mod \ n \mid \ 0 \leq a \leq n-1\}$$

es un sistema de residuos completo [4] [5], es decir, contiene a todos los números entre 0 y $n-1$. Así, hay una correspondencia uno a uno entre los número en el rango 0 y $n-1$ y los números del conjunto anterior. El algoritmo de reducción de Montgomery aprovecha esta propiedad utilizando una rutina de multiplicación más rápida la cual calcula el residuo-n del producto de dos enteros, los cuales sus residuos-n son dados. Dados dos residuos-n $\bar{a}$ y $\bar{b}$ el producto de Montgomery está definido como el residuo-n

$$\bar{c} = \bar{a} \cdot \bar{b} \cdot r^{-1} \ (mod \ n,) \tag{2}$$

donde r^{-1} es el inverso de r mod n con la propiedad

$$r \cdot r^{-1} = 1 \ (mod \ n)$$

El número c, resultado de la ecuación 2, es en realidad el residuo-n del producto $c = a \cdot b \ (mod\ n)$, donde

$$\begin{aligned} \bar{c} &= \bar{a} \cdot \bar{b} \cdot r^{-1} \ (mod\ n) \\ &= a \cdot r \cdot b \cdot r \cdot r^{-1} \ (mod\ n) \\ &= c \cdot r (mod\ n) \end{aligned}$$

Para describir el algoritmo de reducción de Montgomery necesitamos calcular n', el cual es un entero con la propiedad $r \cdot r^{-1} - n \cdot n' = 1$. Los enteros r^{-1} y n', ambos pueden ser calculados con el Algoritmo Extendido de Euclides. Ya que el $mcd(r, n) = 1$, entonces existen dos números r^{-1} y n' con $0 < r^{-1} < n$ y $0 < n' < r$.

El cálculo de $Monpro(\bar{a}, \bar{b})$ realiza con el algoritmo de reducción, el cual se presenta en pseudocódigo en el siguiente cuadro:

Algoritmo 1 -$MonPro(\bar{a}, \bar{b})$-

Entrada: Números enteros $\bar{a}$ y $\bar{b}$.
Salida: El cálculo de $\bar{a} \cdot \bar{b} \cdot r^{-1}$ mod n.

```
1: t := ā · b̄;
2: u := (t + (t · n' mod r) · n)/r;
3: si u > n entonces
4:     u := u − n;
5: fin si
6: devolver u;
```

El algoritmo 1 está basado en el hecho de que el cálculo para $\bar{a} \cdot \bar{b} \cdot r^{-1}$ mod n puede ser hecho eficientemente por algoritmo reducción [3] [2] [6] [1]. Esta rutina requiere de la aritmética módulo r la cual, como se ha comentado, es sencilla de calcular en un sistema de cómputo si $r = 2^k$. Así, el producto de Montgomery es un algoritmo potencialmente más rápido que el cálculo ordinario de $a \cdot b \ mod\ n$ el cual envuelve divisiones por n. Sin embargo, la conversión de un residuo ordinario a un residuo-n, el cálculo de n' y volver a convertir al residuo ordinario consumen gran cantidad de tiempo y no es buena idea usar esto si queremos usar el algoritmo de Montgomery para calcular la multiplicación modular de números muy pequeños. Esto alcanza un buen desempeño para enteros largos.

Se implementa el Algoritmo 1 como el procedimiento $MonPro$ para desarrollar el algoritmo de Montgomery y calcular $c := a \cdot b \ mod\ n$. El Algoritmo 2 se muestra en el siguiente cuadro:

Algoritmo 2 -$montgomery(a, b, n)$-

Entrada: Números enteros a, b y n.
Salida: Multiplicación Modular $c = a \cdot b \ mod\ n$.

```
1: Calcular n' con el Algoritmo Extendido de Euclides;
2: a := a · r mod n;
3: b := b · r mod n;
4: c := MonPro(a, b);
5: c := Monpro(c, 1);
6: devolver c;
```

III. Descripción en Hardware del Algoritmo de Montgomery

La descripción en hardware del algoritmo de Montgomery está compuesta por los módulos Divisor por Restauración (DPR), Multiplicador n bits por m bits (Mnm), Algoritmo Extendido de Euclides (AEE) para calcular n', Dominio de Montgomery (DM), Cálculo de r (Cr) y Algoritmo de Montgomery (AM) el cual contiene el módulo Reducción de Montgomery (RM). Para los resultados que requieran de una precisión mayor a 32 bits, se hace uso del módulo Formato. Cada uno de los módulos anteriores es detallado en las siguientes secciones, describiendo su funcionamiento por medio de diagramas de estados, así como las señales de control y sincronización de cada uno.

A. División por restauranción

El algoritmo de división por restauración (DPR) es un algoritmo eficiente para calcular el cociente y el residuo de divisiones enteras [7]. Así como la multiplicación se puede realizar por medio de una serie de sumas y corrimientos, la división se puede realizar por medio de substracciones y corrimientos. Utilizando estos principios se diseñó el divisor de números enteros sin signo representados con n bits, el cual se muestra en la Figura 1.

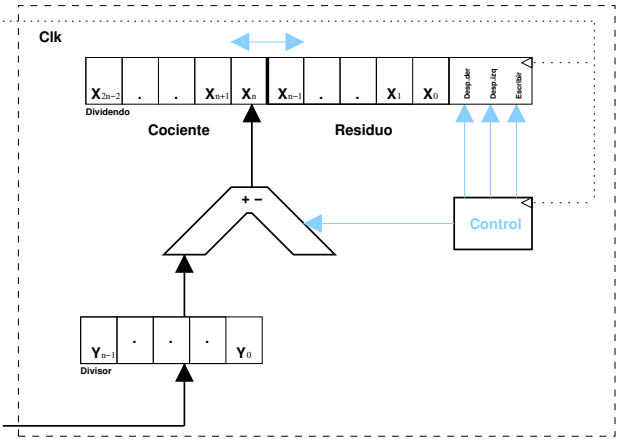

Fig. 1: Circuito del módulo divisor de n bits.

El módulo DPR está formado por un registro de X_{2n-1} bits en el cual se carga el dividendo cuando se requiere hacer una división, un registro de Y_{n-1} bits para cargar el divisor. El divisor cuenta con una unidad aritmética lógica la cual se encarga de hacer operación de substracción y comparación, dependiendo del resultado de la unidad aritmética, el control toma la decisión de hacer un corrimiento a la izquierda, colocando un 1 si la substracción se pudo hacer y un 0 en caso contrario. La unidad de control del circuito divisor es implementado en base al diagrama de estados mostrado en la Figura 2.

El funcionamiento de la unidad de control del DPR es el siguiente: en el estado S_0, si la señal $St = 1$, se habilita la señal de carga de operandos $Ld = 1$ e inician las operaciones, de lo contrario el circuito está inactivo. En el estado S_1, si $C_i = 1$ indica que el cociente requerirá más bits, entonces

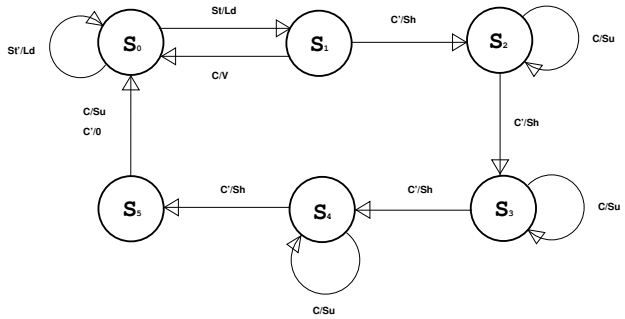

Fig. 2: Máquina de estados para la división por restauración.

$V = 1$ y se suspenden las operaciones, de lo contrario se realiza un corrimiento a la izquierda $Sh = 1$. En los estados S_2, S_3 y S_4, si $C = 1$, se realiza la substracción $Su = 1$ y se realiza el corrimiento a la izquierda $Sh = 1$ pasando al siguiente estado. En S_5, si $C = 1$, se realiza la substracción $Su = 1$ y se pasa al estado final.

B. Multiplicación serial

Fue necesario diseñar un circuito multiplicador de n (multiplicador) por m (multiplicando) bits para poder multiplicar enteros muy grandes, por ejemplo de 1024 bits. El algoritmo utilizado sigue los principios básicos del algoritmo para multiplicar números en base 10 [7], como se mencionó anteriormente, la multiplicación se puede llevar acabo de manera sencilla a través sumas y corrimientos. Es importante aclarar que el módulo multiplicador solo puede multiplicar números sin signo.

El módulo multiplicador está formador por registros de almacenamiento y corrimiento, unidades aritméticas para la suma y comparación y su unidad de control. El diagrama del circuito multiplicador es similar al circuito divisor mostrado en la Figura 1, sin embargo el control sigue un diagrama de estados diferente, el cual se muestra en la Figura 3.

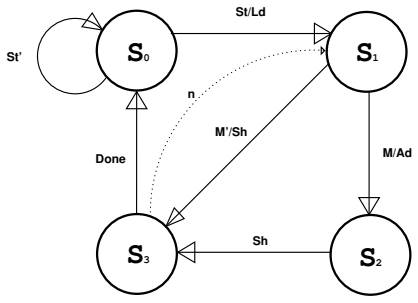

Fig. 3: Máquina de estados para la multiplicación

En la figura anterior se puede observar que la multiplicación se lleva a cabo en los estados S_1, S_2 y S_3, los cuales representan un ciclo que se detiene hasta llegar a n. En el estado S_0 se verifica si está activado el circuito St, si lo está, entonces se cargan los operandos mediante la señal Ld y se procede a calcular la operación con los operandos. En S_1 se verifica una bandera M, si está activada pasa al estado S_2, además de calcular una suma Ad; de lo contrario pasa al estado S_2 con un corrimiento a la derecha Sh. En el estado S_2 simplemente se hace un corrimiento más a la derecha y pasa al estado S_3. En el estado S_3 verifica si se han cumplido el número de operaciones, dependiendo de la precisión de los registros para efectuar la multiplicación, si se cumplen pasa al estado S_0 terminando así de calcular la multiplicación encendiendo una bandera $Done$, de lo contrario regresa al estado S_1 y continua hasta terminar el cálculo.

C. Algoritmo Extendido de Euclides

El Algoritmo Extendido de Euclides (AEE) es utilizado para calcular los inversos modulares. Para la implementación en hardware del AEE se utilizó el módulo Divisor por restauración (DPR). El DPR calcula cocientes y residuos mientras la condición impuesta por el AEE se cumpla. El diagrama de estados del AEE se presenta en la Figura 4 y en la sección donde se explica el Coprocesador, se presenta su correspondiente implementación en hardware.

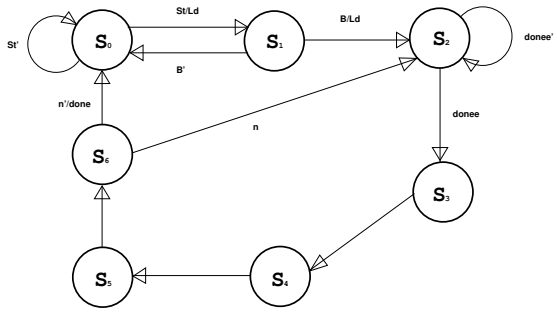

Fig. 4: Máquina de estados para el cálculo de n' por medio del Algoritmo Extendido de Euclides

En el estado S_0, si $St = 1$, se cargan los operandos activando $Ld = 1$ e inician las operaciones, de lo contrario el circuito queda inactivo. En el estado S_1, si $B = 1$, indica que el cociente requerirá más bits suspendiendo las operaciones, de lo contrario se pasa al siguiente estado. En el estado S_2 se habilita el divisor y se obtiene el cociente y residuo necesarios para pasar a los siguientes cálculos en el siguiente estado. En el estado S_3 se realizan dos multiplicaciones y en los estados S_4 y S_5 se calculan sumas y asignaciones. En el estado S_6, si $n > 0$, se continua con la ejecución de la pasando al estado 2, de lo contrario se habilita la señal $done = 1$ y pasa al estado final.

D. Dominio de Montgomery

El módulo Dominio de Montgomery (DM) pasa los operandos a y b a su respectivo Dominio de Montgomery, lo que implica calcular el residuo-n de cada operando. Para la implementación de este módulo en hardware también se utiliza el módulo Divisor por restauración (DPR) y del módulo Cálculo de r (Cr), el cual realiza corrimientos a izquierda para multiplicar los operandos por r. El diagrama de estados del módulo Dominio de Montgomery se presenta en la Figura 6 y su funcionamiento es el siguiente: en el estado S_0, si $St = 1$, se cargan los operandos con la señal $Ld = 1$ e inician las operaciones calculando a r, de lo contrario el circuito está

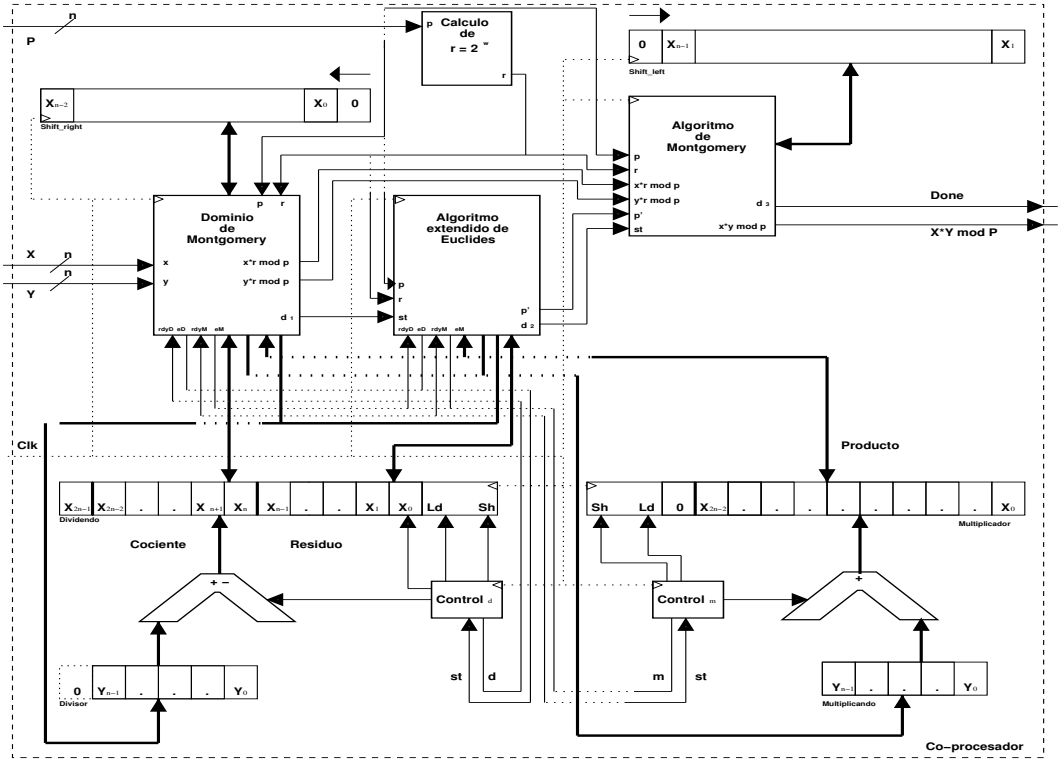

Fig. 5: Ruta de datos y control del Coprocesador Matemático de Aritmética Entera (CMAE).

inactivo. En el estado S_1, se calculan las multiplicaciones por r mediante corrimientos a la izquierda. En los estados S_2 y S_4 se activa el divisor y se obtiene el residuo, $donee = 1$ y pasa al estado final.

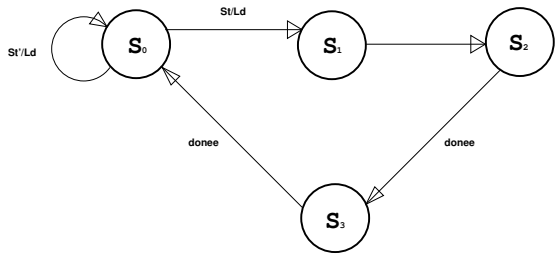

Fig. 6: Máquina de estados para el cálculo de de los residuos-n de a y b

E. Reducción de Montgomery

El módulo Algoritmo de Montgomery (AM) contiene las operaciones para realizar la Reducción de Montgomery (RM) para calcular $c = MonPro(\bar{a}, \bar{b})$, y se implementa en base al pseudocódigo mostrado en el cuadro 1. Para ello se requiere previamente calcular a r, los residuos-n de los operandos a y b y el inverso modular n' a través del Algoritmo Extendido de Euclides. El diagrama de estados del módulo Reducción de Montgomery se observa en la Figura 7.

En el estado S_0, si $St = 1$ se cargan los operandos con la señal $Ld = 1$ e inician las operaciones. En los estados S_1 S_2 y S_3 se calcula t y u, es decir, $c = MonPro(a', b')$. En el estado S_4, si $B = 0$ se regresa al estado S_2 y se calcula

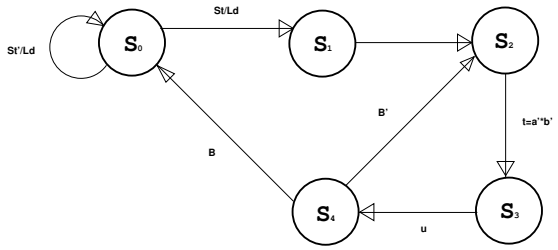

Fig. 7: Máquina de estados para el cálculo de $a \cdot b \cdot r^{-1}\ mod\ n$ con ayuda del algoritmo de reducción de Montgomery

$c = MonPro(c, 1)$, de lo contrario se pasa el estado final $B = 1$.

IV. Coprocesador Matemático de Aritmética Entera

El Coprocesador Matemático de Aritmética Entera (CMAE), incluye todos los módulos mencionados en la sección anterior y su arquitectura completa se muestra en la Figura 5. El CMAE recibe como entrada tres valores: X, Y y P, éstos son números enteros de n de bits y como salida una señal de terminación de la operación y un vector con el resultado: $X * Y\ mod\ n$. Para pasarle los operandos al coprocesador a través del sistema de buses, es necesario dividir X, Y y P en palabras de 32 bits y almacenarlos en los registros respectivos.

Como se mencionó en la explicación del Algoritmo de Montgomery, para que el coprocesador funcione de manera correcta P debe ser un número primo, X y Y deben de ser números enteros positivos menores que P. Además recibe una

señal rdy de parte del divisor, esta señal le indica que se ha terminado la división y que el módulo puede prescindir de los resultador del divisor, también tiene una señal e la cual habilita al divisor para cargar sus operandos y comenzar los cálculos.

El coprocesador cuenta con un circuito combinatorio para calcular $r = 2^w$, el cual recibe un vector P como entrada, de tal manera que $mcd(r, p) = 1$. Una vez calculado a r se activa el módulo DM para calcular los residuos-n de los operandos de entrada X y Y. Este módulo recibe como entrada X, Y números enteros, un número primo P, el valor de r. Como salida se obtienen $x * r \ mod \ n$, $y * r \ mod \ n$ y una señal d_1 de habilitación del módulo AEE para calcular el inverso modular. El módulo DM depende del módulo DPR y un registro de n bits para poder hacer corrimientos a la izquierda.

El módulo AEE recibe como entrada a P, r y su señal de habilitación st. Esta señal de inicio es enviada por el módulo DM la cual le indicá que puede activarse y comenzar las operaciones. También son enviadas las señales rdy y e para el control del divisor. Como salida se obtiene p' el cual es el inverso modular de P.

El módulo AM recibe como entrada, los valores calculados anteriormente por los otros módulos, es decir; recibe al vector P, $r = 2^w$, $x * r \ mod \ n$, $y * r \ mod \ n$, el inverso modular p' una señal de habilitación st, la cual se activa, con la señal de culmincación d_2, del módulo que calcula el inverso modular p'. Además el módulo está comunicado con un registro que hace corrimientos a la derecha y así realizar divisiones de manera más rápida. Como salida se obtiene el vector $x * y \ mod \ n$, este vector contiene la operación hecha por el coprocesador utilizando la multiplicación modular del algoritmo de Montgomery y una señal $Done$ de culminación de las operaciones.

El DPR tiene una señal como entrada st, la cual está comunicada con los módulos de DM y AEE, ésto habilitará al DPR cuando prescindan del cociente y el residuo de la división. El control decide si hacer un corrimiento a la izquierda modificando el bit menos significativo del registro X_0 o simplemente hacer el corrimiento sin ninguna modificación, además el control también es quien se encarga de cargar los operandos, dividendo y divisor. Cuando se ha terminado de hacer la división el control manda la señal d_0 al módulo que hizo la solicitud de dividir dos núemeros, podrá obtener el cociente y el residuo del registro, siendo $X_{2n-2} - X_n$ el resultado del cociente y $X_{n-1} - X_0$ para el residuo.

Los registros del DPR están comunicado con el módulo DM y se encarga de hacer las divisiones a través de corrimientos a la derecha, de manera más rápida y eficiente. El módulo AEE tiene comunicación con otro registro para hacer corrmientos $shift\_right$; se encarga de hacer las multiplicación, a través de corrimientos a la izquierda. Para ambos registros se indica el número de corrimientos que se harán, una vez realizado el el corrimiento, el registro vuelve a cargar el vector de resultados al módulo que hizo la solicitud.

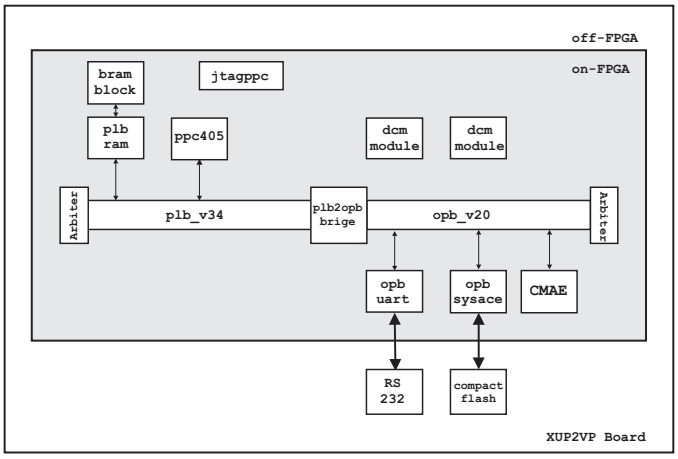

Fig. 8: CMAE en un sistema embebdido basado en un FPGA.

V. Sistema en un Chip

La tarjeta desarrollo XUPV2P (*Xilinx University Program, Virtex II Pro*) [8], es parte del proyecto de colaboración entre la compañía Xilinx y las universidades, el cual tiene como propósito acercar a los estudiantes y académicos a la tecnología de los circuitos programables, como los FPGAs y sus herramientas de desarrollo. La tarjeta cuenta con un FPGA Virtex-II Pro (XC2VP30) y dispositivos para el desarrollo de aplicaciones, destacando 10/100Mbps Ethernet PHY, una ranura para una tarjeta Compact Flash, dos ranuras para DIMMs de memoria DDR-SRAM, y un puerto RS-232, entre muchos otros. El FPGA XC2VP30 además de contar con una gran densidad de CLBs (*Configurable Logic Block*) para la implementación de circuitos digitales, tiene incrustados dos *hard processors core* PowerPC-405 y varios bloques de SRAM que en conjunto suman 2 MBytes.

Mediante los CPUs y los bloques de SRAM incrustados en el FPGA, es posible cargar y ejecutar programas desarrollados en un lenguaje de alto nivel, como C. Dichos programas deben ser compilados, ensamblados y enlazados previamente por las herramientas de desarrollo, para después ser cargados en los bloques SRAM en el momento de integrar el hardware y software. El programa será ejecutado inmediatamente después de la programación del FPGA.

En el FPGA XC2VP30 se creó el sistema de cómputo embebido mostrado en la Figura 8, el cual está compuesto del procesador incrustrado (*hard core*) PowerPC 405 a 400Mhz (`ppc405`), un bloque de memoria SRAM *On-Chip* de 128KBytes (`bram_block`) con su respectivo módulo de acceso al bus (`plb_ram`), un sistema de buses jerárquico compuesto del bus local del procesador (`plb`), del bus de periféricos en chip (`opb`) junto con sus respectivos árbitros y un puente entre buses (`plb2opb`). Para poder enviar los números y almacenar los resultados fue necesario agregar el módulo (`opb_sysace`) para el acceso a un sistema de archivos FAT12 sobre *Compact Flash Memory* de 1GBytes. Se incluyó un transmisor-receptor serial para la entrada y salida estándar del sistema (`opb_uart`). El CMAE fue agregado como un circuito periférico y por medio del bus `opb` el

programa que se ejecuta en el PowerPC podrá direccionarlo para enviar y recibir datos del mismo.

VI. Resultados

Para probar el funcionamiento correcto del CMAE se realizaron varios experimentos con números de 512 bits y de 1024 bits. Los experimentos consistieron en realizar la operación $(a \cdot b \ mod \ p)$ utilizando MAPLE y el CMAE, para después compararlos.

El resultado de la multiplicación modular $(a \cdot b \ mod \ p)$ con números de 512 bits, es también un número de 512 bits. En la Tabla I se muestran los valores de los de los operandos y el resultado de la operación expresado en dígitos hexadecimales.

TABLA I: Operandos y resultados con números de 512 bits.

a_{16}	b_{16}	p_{16}	$(a \cdot b \ mod \ p)_{16}$
12345678	98765432	5D54A5F6	286A0277
99876543	11236547	EE7D8AF7	2562A88A
21123456	89987654	11FDE83A	375A913C
78998765	32112345	E16B4DD0	785D3D14
43211234	67899876	EE26A39F	07C9A824
56789987	54321123	DA62B2EF	0E505B93
65432112	45678998	AB5C1ED3	C343419C
34567890	76543210	4F912E6B	245231C0
54321AB0	98765432	B8E54C43	E8F23147
1025BF05	1BBBB00C	2FCDA6D2	7E50CF1C
012340AB	C1234567	E2A3D605	95ACC9E8
1020BF05	8B4512AA	A0C66215	8488C252
C1234000	0B4512AA	5B4919A4	F977AECF
00000000	A000000C	FCA3EAA0	DE218CB3
C12345AB	C1234567	4D32CA00	781D1949
1025BF05	8B4512AA	00000001	95A36FDA

Se pudo verificar que los resultados arrojados por el CMAE son exactamente los mismos que se obtuvieron al utilizar MAPLE. En el segundo experimento se utilizaron números a, b y p (número primo) de 1024 bits, escalándo el CMAE a 1024 bits. De igual forma se obtuvieron resultados exactos entre MAPLE y el CMAE.

A. Tiempo de ejecución

Se realizaron pruebas de tiempo de ejecución únicamente para números de 512 y 1024 bits. La Tabla II muestra la realación en tiempos de ejecución medido en segundos para las diferentes versiones probadas: programa realizado en lenguaje C, ejecución en Maple y el tiempo por el CMAE.

TABLA II: Tabla comparativa de tiempo de ejecución

Bits	Programa en C	CMAE	MAPLE
512	1.376	0.003229510	0.06
1024	5.814	0.012621650	0.07

VII. Conclusiones

Para escalar el coprocesador para números muy largos, por ejemplo de 256, 512, 1024 o 2048 bits, se necesita la base del coprocesador para números de 32 bits. Partiendo de esta base es posible escarlo hasta donde los recursos del FPGA lo permita. Una estratégia de escalamiento, es dividir los operandos en palabras de 32 bits formando arreglos de palabras de 32 bits que representen los números de n bits. El coprocesador alcanzará su máximo desempeño y eficiencia si los operandos con los cuales se desea realizar la multiplicación modular son muy grandes. Esto se debe a que las operaciones necesarias para realizar la multiplicación modular de Montgomery cuando son implementadas en hardware, se convierten en operaciones sencillas, mientras que las mismas operaciones son muy costosas en tiempo cuando son implementadas en software.

Referencias

[1] Çetin Kaya. Koç, *Cryptographic Engineering*. Springer, 2009.
[2] A. F. Tenca and Çetin Kaya Koç, "Montgomery algorithm for modular multiplication," *IEEE Micro*, vol. 52, pp. 1215–1221, 2003.
[3] D. J. Guan, "Montgomery algorithm for modular multiplication," *Department of Computer of Science, National Sun Yar-Sen Univeristy, Kaohsiung, Taiwan*, 2003.
[4] H. S. Z. Niven I., *Introducción a la teoría de números*. Limusa, 1976.
[5] R. Johnsonbaugh, *Matemáticas Discretas*. Richard Johnsonbaugh, 2005.
[6] Çetin Kaya Koç and T. Acar, "Analyzing and comparing montgomery multiplication algorithms," *IEEE Micro*, vol. 16, pp. 26–33, 1996.
[7] D. A. Patterson and J. L. Hennessy, *Computer Organization and Design*. Morgan Kaufmann, 2009.
[8] (2012, Jan.) Xupv2p documentation. [Online]. Available: http://www.xilinx.com/univ/xupv2p.html

Joel Noyola Bautista Actualmente es ayudante de profesor en docencia e investigación en el Departamento de Sistemas de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco. Su interés se ha centrado en el desarrollo de aplicaciones criptográficas en hardware.

Oscar Alvarado-Nava Actualmente es Profesor-Investigador Titular en el Departamento de Electrónica de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco.

Felipe Monroy Pérez Actualmente es Profesor-Investigador Titular en el Departamento de Ciencias Básicas de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco.