Descripción con VHDL de un exponenciador para campos finitos $GF(2^m)$

Mario Alberto García Martínez[1], *Guillermo Morales Luna*[2]
y Francisco Rodríguez Enríquez[2]
(1) Universidad del Valle de México, campus Lomas Verdes
Paseo de las Aves No. 1, Col. San Mateo Nopala
53220, Naucalpan de Juárez, Edo. de México
correo-e: marioag@prodigy.net.mx

(2) Sección de Computación, CINVESTAV, IPN
Av. IPN 2508, Col. San Pedro Zacatenco
07300, México D.F.
correo-e: gmorales@cs.cinvestav.mx ;
francisco@delta.cs.cinvestav.mx

ABSTRACT

Exponentiation in finite or Galois fields, $GF(2^m)$, is a basic operation for several algorithms in areas such as cryptography, error-correction codes and digital signal processing. Nevertheless the involved calculations are very time consuming, especially when they are performed by software. Due to performance and security reasons, it is rather convenient to implement cryptographic algorithms by hardware. In order to overcome the well-known drawback of reduced flexibility associated to traditional ASIC solutions, we propose an architecture using field programmable gate arrays (FPGA´s). A cheap and flexible modular exponentiation can be implemented using these devices.
We introduce the VHDL description and implementation of an architecture for exponentiation in $GF(2^m)$ based in the square-and-multiply method, called *binary method*, using two multipliers previously developed by ourselves. Our structure, compared with other structures reported earlier, introduces an important saving in hardware components.

RESUMEN

La operación de exponenciación en campos finitos o de Galois $GF(2^m)$ es de considerable importancia en el desarrollo de algoritmos en las áreas de criptografía, códigos de corrección de errores y procesamiento digital de señales. Sin embargo los cálculos que se requieren consumen una gran cantidad de tiempo sobre todo cuando son realizados en software. Por razones de desempeño y seguridad, es preferible implementar tales algoritmos en hardware. Con el propósito de evitar el inconveniente de la poca flexibilidad que implican las soluciones de diseño con ASIC´s, proponemos una arquitectura para FPGA´*s (Field Programmable Gate Array*). Una exponenciación económica y flexible puede ser implementada con el uso de estos dispositivos.
Se presenta aquí la descripción en VHDL y la implementación de una estructura para el cálculo de la exponenciación en $GF(2^m)$ basada en el método de elevación al cuadrado y multiplicación, también conocido como *método binario*, que usa dos multiplicadores seriales que hemos desarrollado previamente. Nuestra estructura, comparada con otras publicadas anteriormente, presenta un considerable ahorro en recursos de hardware.

Descripción con VHDL de un exponenciador para campos finitos $GF(2^m)$

Mario Alberto García Martínez[1], *Guillermo Morales Luna*[2]
y Francisco Rodríguez Enríquez[2]
(1) Universidad del Valle de México, campus Lomas Verdes
Paseo de las Aves No. 1, Col. San Mateo Nopala
53220, Naucalpan de Juárez, Edo. de México
correo-e: marioag@prodigy.net.mx

(2) Sección de Computación, CINVESTAV, IPN
Av. IPN 2508, Col. Zacatenco
07300, México D.F.
correo-e: gmorales@cs.cinvestav.mx ;
francisco@delta.cs.cinvestav.mx

Resumen

La operación de exponenciación en campos finitos ó de Galois $GF(2^m)$ es de considerable importancia en el desarrollo de algoritmos en las áreas de criptografía, códigos de corrección de errores y procesamiento digital de señales. Sin embargo los cálculos que se requieren consumen una gran cantidad de tiempo sobre todo cuando son realizados en software. Por razones de desempeño y seguridad, es preferible implementar tales algoritmos en hardware. Con el propósito de evitar el inconveniente de la poca flexibilidad que implican las soluciones de diseño con ASIC´s, proponemos una arquitectura para FPGA´s (*Field Programmable Gate Array*). Una exponenciación económica y flexible puede ser implementada con el uso de estos dispositivos.

Se presenta aquí la descripción en VHDL de una estructura para el cálculo de la exponenciación en $GF(2^m)$ basada en el método de elevación al cuadrado y multiplicación, también conocido como *método binario*, que utiliza dos multiplicadores seriales que hemos desarrollado previamente. Nuestra estructura, comparada con otras publicadas anteriormente, presenta un considerable ahorro en recursos de hardware.

I. Introducción

La exponenciación sobre campos finitos o de Galois es fundamental en varios algoritmos criptográficos de uso generalizado actualmente, tales como el método de Diffie-Helman para el intercambio de llaves [1], el algoritmo de El-Gamal para firmas digitales [2] o el criptosistema RSA [3]. Los cálculos requeridos para tales algoritmos implican un alto consumo de tiempo de proceso, especialmente cuando son realizados por software. Un método convencional para el cálculo de exponenciaciones en campos finitos por software hace uso de tablas de consulta. Pero este método no se puede realizar eficientemente en un circuito del tipo VLSI (*Very Large Scale of Integration)*. Por razones de desempeño y de seguridad, es preferible desarrollar los algoritmos criptográficos en hardware. Por otro lado, con el propósito de evitar la poca flexibilidad asociada al diseño de los tradicionales ASIC´s (*Application Specific Integrated Circuit*), proponemos una arquitectura para FPGA´s (*Field Programmable Gate Array*). Usando tales dispositivos se puede implementar un exponenciador económico y flexible.

El método de elevación al cuadrado y multiplicación, conocido como *método binario* [4], es una técnica aceptada generalmente para el cálculo de una exponenciación, digamos $R = M^e$, en campos finitos. Existen dos versiones para este algoritmo: *MSB-first* y *LSB-first (Most and Least Significant Bit,* respectivamente). Cada una depende de cuál es el primer bit del exponente que se examina en el algoritmo. En este trabajo hemos utilizado la versión *LSB-first* ya que nos permite el desarrollo de una estructura que operará en paralelo el cálculo de la exponenciación. Hacemos uso de dos multiplicadores diseñados en [5]. Si comparamos nuestra estructura con las presentadas en [6] y [7] apreciaremos el ahorro en hardware que se obtiene. Presentamos la estructura y su descripción en VHDL (*VHSIC Hardware Description Language*, VHSIC a su vez es *Very High Scale Integration Circuits*). Posteriormente hemos de implementarlo físicamente en un FPGA.

II. Algoritmo de exponenciación

Sea M un elemento arbitrario en $GF(2^m)$ expresado como:

$$M = \sum_{i=0}^{i=m-1} m_i \alpha^i$$

y sea e, $(1 \le e \le 2^m\text{-}1)$ un entero cuya representación binaria es:

$$e = \sum_{i=0}^{n-1} e_i 2^i = (e_{n-1}, e_{n-2}, ..., e_1, e_0); e_i = \{0,1\}$$

La potencia $R = M^e$ está también en $GF(2^m)$ y, según el método binario [4], se calcula mediante el:

Algoritmo: *(Exponenciación LSB-first)*

Input: *M, e, G*
Output: $R = M^e \pmod G$
==========
1.- $C := M$; $R := 1$;
2.- **for** $i := 0$ **to** $n\text{-}1$ **do**
2.*a*).- **if** $e_i := 1$ **then** $R := R*C \pmod G$
2.*b*).- $C := C*C \pmod G$
end for ;
3.- **return** *R*;
==========

Ejemplo 1
$e = 0\ 1\ 1\ 1\ 0\ 0\ 1\ 0 = 114$

e	***Step 2.a (R)***	***Step 2.b (C)***
0	1	$(M)^2 = M^2$
1	$1*M^2 = M^2$	$(M^2)^2 = M^4$
0	M^2	$(M^4)^2 = M^8$
0	M^2	$(M^8)^2 = M^{16}$
1	$M^2 * M^{16} = M^{18}$	$(M^{16})^2 = M^{32}$
1	$M^{18} * M^{32} = M^{50}$	$(M^{32})^2 = M^{64}$
1	$M^{50} * M^{64} = M^{114}$	$(M^{64})^2 = M^{128}$
0	M^{114}	$(M^{128})^2 = M^{256}$

Ejemplo 2
$e = 1\ 1\ 1\ 1\ 1\ 0\ 1\ 0 = 250$

e	***Step 2.a (R)***	***Step 2.b (C)***
0	1	$(M)^2 = M^2$
1	$1*M^2 = M^2$	$(M^2)^2 = M^4$
0	M^2	$(M^4)^2 = M^8$
1	$M^2 * M^8 = M^{10}$	$(M^8)^2 = M^{16}$
1	$M^{10} * M^{16} = M^{26}$	$(M^{16})^2 = M^{32}$
1	$M^{26} * M^{32} = M^{58}$	$(M^{32})^2 = M^{64}$
1	$M^{58} * M^{64} = M^{122}$	$(M^{64})^2 = M^{128}$
1	$M^{122} * M^{128} = M^{250}$	$(M^{128})^2 = M^{256}$

III. Arquitectura del exponenciador

El diagrama de flujo del exponenciador se muestra en la Fig.1. En la Fig.2, proponemos una arquitectura paralela para la exponenciación en $GF(2^m)$ basada en el método binario. Esta estructura requiere de n multiplicaciones y sn^2 ciclos de reloj para el cálculo de una exponenciación modular en la que el exponente es un número de n-bits y s es el número de ciclos de reloj requeridos para calcular la multiplicación. En nuestro caso, usamos un multiplicador sistólico y serial (*SSM*) [5] cuyo tiempo de retardo es de $3m$-1 ciclos de reloj.
La arquitectura utiliza dos multiplicadores *SSM* que trabajan en paralelo, dos registros R y C de m-bits y un multiplexor que selecciona la operación correspondiente de *SMM(1)*.

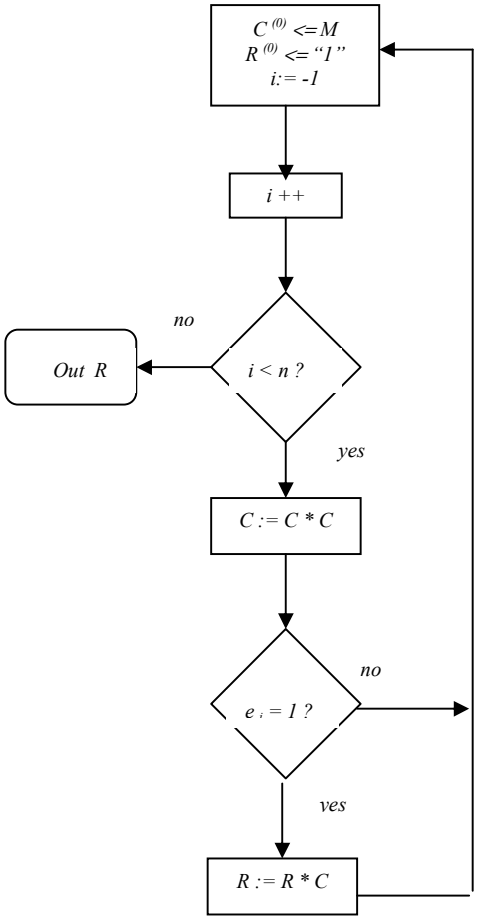

Fig.1 *Diagrama de flujo del método binario*

Como se puede observar en el algoritmo, los registros R y C son cargados inicialmente con "1" y

M respectivamente. Entonces, con cada ciclo de reloj, *SMM(2)* operará para calcular *C*C*; y, dependiendo del valor de e_i, *SMM(1)* trabajará para realizar el producto de *R*C* cada vez que e_i ="1". El resultado final se obtendrá en el registro *R* cuando el bit *n*-1 de *e* sea examinado.

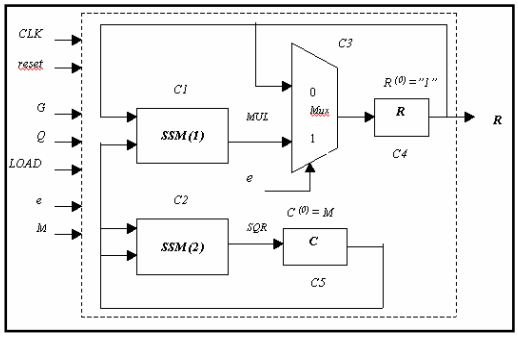

Fig.2. *Arquitectura del Exponenciador*

En la estructura, las señales son:

SSM=*Multiplicador sistólico y serial*
Mux=*Multiplexor*
R,C=*Registros con carga paralela*
G = *Modulo,* **Q** = *Señal de control*

Este circuito se especifica directamente en VHDL. Su código lo presentamos en el Apéndice A.

IV. Implementación

Posterior a la síntesis del circuito, se ha implementado un exponenciador de 32 bits en un FPGA Virtex XSV300 de Xilinx. Se presenta en las Figuras 3 y 4 el Layout del circuito.

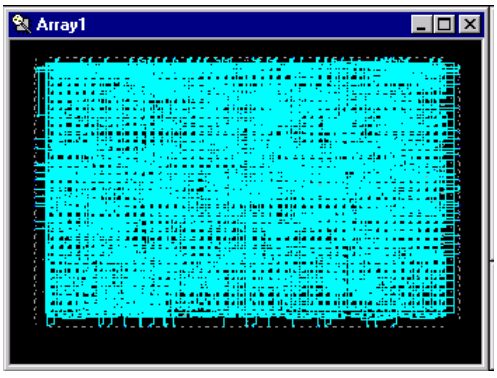

Fig.3. *Layout de conexiones en el FPGA*

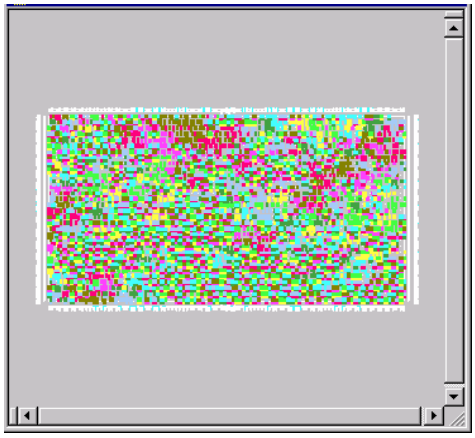

Fig.4. *Layout de la distribución de CLB´s en el FPGA*

V. Resultados

En la tabla 1 se muestra el consumo de CLB´s en el FPGA utilizado por el exponenciador de 32 bits.

```
Device utilization summary:

   Number of External GCLKIOBs         1 out of 4      25%
   Number of External IOBs            86 out of 260    33%
      Number of LOCed External IOBs    0 out of 86      0%

   Number of SLICEs                2972 out of 3072    96%

   Number of GCLKs                    1 out of 4      25%
```

Tabla 1. *Consumo de recursos del exponenciador de 32 bits.*

La tabla 2 muestra las exigencias de hardware de nuestro exponenciador y su comparación con [6] y [7].

	[6]	[7]	Aquí
Multiplicaciones	2(*m*-1)	2(*m*-1)	*n*
Multiplicadores	*m*-1	2(*m*-1)	2
Elevadores al cuadrado	*m*-1	------	--
Registros	----	------	2
Multiplexores	*m*	------	1
Tiempo de cómputo	m^2-*m*+*m*/2+1	$2m^2$ +2	$(3m-1)n^2$

Tabla 2. *Exigencias de hardware y tiempos para la exponenciación*

Como puede observarse de la tabla, en [6] y [7] hacen depender toda la estructura del valor de la dimensión del campo *m*, que es la longitud de las

palabras en el campo, mientras que en el diseño que presentamos la dependencia es principalmente de *n*, que es la longitud del exponente *e*.
Para la descripción VHDL y la implementación se han utilizado las herramientas del programa ISE 4.1i de Xilinx, para lo que se cuenta con una tarjeta prototipo cuyo FPGA es un XSV300 de la serie Virtex de Xilinx el cual está integrado con 3072 CLB´s (*Configurable Logic Blocks*).
En este momento nos encontramos realizando las pruebas de campo al circuito, y posteriormente se hará un proceso de optimización.

VI. Conclusiones

Hemos presentado la arquitectura y la descripción VHDL de un exponenciador para campos finitos que se basa en el método binario en su versión *LSB-first*. Es una estructura que opera en paralelo y que es económica en cuanto a sus necesidades de hardware. Se ha implementado un exponenciador de 32 bits en un FPGA al que se le realizan las pruebas de campo en este momento. El exponenciador ha de utilizarse como elemento fundamental en algoritmos criptográficos y en códigos de corrección de errores.

VII. Referencias

[1] Diffie,W. and Hellman, M.E. : "New directions in cryptography", *IEEE Trans. Inf. Theory, 1976*, IT-22,(6), pp.644-654.

[2] ElGamal, T.: "A public-key cryptosystem and a signature scheme based on discrete logarithms". *IEEE Trans. Inf. Theory*, 1985, (31), pp.469-472.

[3] Rivest, R.L., Shamir,A. and Adleman,L.: "A method for obtaining digital signatures and public-key cryptosystems*", Commun. ACM*, 1978, 21, (2), pp.120-126.

[4] Knuth, D.E. *: "The art of computer programming, vol.II: Seminumerical algorithms*", (Adison-Wesley, MA, 1969).

[5] García-Martínez,M.A. and Morales-Luna G. : "FPGA implementation to divide and multiply in GF(2^m)". Finite Fields and Aplications in Error Correcting Codes and other areas. *LNCS Springer-Verlag*, 2002.

[6] Jain, S.K., Song, L. and Parhi, K.K. : "Efficient semisystolic architectures for finite field arithmetic". *IEEE Trans. on VLSI*,1998, vol.6 (1),pp. 101-113.

[7] Wang, C.L. "Bit-Level Systolic Array for Fast Exponentiation in GF(2^m)", *IEEE Trans. on Comp.,*1994, vol.43(7), pp. 838-841.

A. Descripción en VHDL del exponenciador

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity exponentiator is                --Main entity
   Port ( M,e,G,Q : in std_logic;
          R : out std_logic;        --R=M^e (mod G)
         CLK,reset : in std_logic);
end exponentiator;

architecture Behavioral of exponentiator is

       component SSM port(
            CLK,reset,G,Q,A,B: in std_logic;
            C: out std_logic);
       end component;

       component Mux21 port(
            X,Y,e: in std_logic;
            Z: out std_logic);
       end component;

        component REG port(
            CLK,reset,LOAD,In_reg: in std_logic;
            Out_reg: out std_logic);
        end component;

signal S_reg_C,S_reg_R,S_SQR,S_MUL,S_In_R:
       std_logic;

begin

 C1:SMM port map
     (CLK=>CLK,reset=>reset,G=>G,Q=>Q,
     A=>S_reg_R,B=>S_reg_R,C=>S_MUL);
 C2: SMM port map
     (CLK=>CLK,reset=>reset,G=>G,Q=>Q,
     A=>S_reg_C,B=>S_reg_C,C=>S_SQR);
 C3: Mux21 port map(X=>S_reg_R,Y=>S_MUL,Z=>In_R);
 C4: REG port map
     (CLK=>CLK,reset=>reset,LOAD=>LOAD,
     In_reg=>S_In_R,Out_reg=>S_reg_R);
C5: REG port map
     (CLK=>CLK,reset=>reset,LOAD=>LOAD,
     In_reg=>S_In_R,Out_reg=>S_reg_R);

end Behavioral;

==========================================

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity REG is                          --Register entity
   Port ( CLK,set,reset,In_LOAD : in std_logic;
          Din_p: in std_logic_vector(m-1 to 0);
          Din_s: in std_logic;
          Dout: out std_logic);
end REG;
```

```
architecture behavior of REG is

signal Q_temp: std_logic_vector(m-1 down to 0);

begin

    Dout<=”0”;

    comb:process(In_LOAD,Din_s)
    begin
            if(In_LOAD=”1”) then Q_temp<=Din_p;end if;
    end process comb;

   state: process(CLK,set,reset)
   begin
      if(reset=”1”) then Q_temp<=(others=>”0”);end if;
      if(set=”1”) then Q_temp<= (others=>”1”);
      elsif(CLK’event and CLK=”1”) then
            Q_temp:=Din_p & Q_temp(m-1 down to 1);
   end if;
      Dout<= Q_temp(0);

end process state;

end behavior;

==========================================

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Mux21 is                    --Mux entity
   Port ( X,Y,e : in std_logic;
              Z: out std_logic);
end Mux21;

architecture behavior of mux21 is

begin
            Z<=Y when (e=”1”) else X;
end behavior;

==========================================
```

Nota: La descripción VHDL del SSM puede consultarse en [5].