Descripción en VHDL de circuitos aritméticos para campos finitos GF(2^m)

Presenta: Mario Alberto García-Martínez

1

Contenido

- Introducción
- Generalidades
- El exponenciador para GF(2^m)
- El multiplicador sistólico y serial
- El multiplicador LSB-first
- El multiplicador por dígitos
- Conclusiones

2

Introducción

- Las **operaciones aritméticas** sobre GF(2^m) son intensamente usadas en criptografía, en códigos de corrección de errores y procesamiento digital de señales
- Esto presenta la necesidad de que tales operaciones puedan ser desarrolladas a **altas velocidades**.

3

- Frente a los desarrollos en software de estos algoritmos, proponemos en este trabajo la **implementación en hardware** de tales operaciones básicas.
- Usamos VHDL para la descripción de los circuitos y usamos las herramientas del paquete ISE4.1i de Xilinx para realizar la síntesis y la simulación de los diseños.

4

Generalidades

- Un campo finito $GF(2^m)$ tiene 2^m elementos, y es una extensión del campo $GF(2)$ ={0,1}.
- Todo campo finito tiene asociado por lo menos un polinomio irreducible de orden m de la forma:
  - $p(x)= x^m + \sum_{i=0}^{m-1} p_i\ x^i$ con p_i= {0,1}

5

Generalidades

- Todo campo finito tiene un elemento primitivo a que es una raíz de $p(x)$ y los elementos del campo pueden ser expresados como potencias de a :
  - $GF(2^m)=\{0, \alpha^1, \alpha^2, \ldots, \alpha^{2m-1}=1\}$
- Y ya que :
  - $p(a)=0$ se tiene que $a^m = P(a)$

6

Generalidades

- Es decir, los elementos del campo pueden ser representados como polinomios en a de orden menor que m operando sobre *GF(2).*
- Así un elemento *A* puede ser expresado como:
  - $A(a) = \sum_{i=0}^{m-1} a_i\, a^i \quad con \quad a_i = \{0,1\}$

7

Considere el campo GF(2^4) cuyo polinomio irreducible es $p(x) = x^4 + x^3 + 1$

Potencia de α	Representación polinomial
α^0	1
α^1	α
α^2	α^2
α^3	α^3
α^4	$\alpha^3 + 1$
α^5	$\alpha^3 + \alpha + 1$
α^6	$\alpha^3 + \alpha^2 + \alpha + 1$
α^7	$\alpha^2 + \alpha + 1$
α^8	$\alpha^3 + \alpha^2 + \alpha$
α^9	$\alpha^2 + 1$
α^{10}	$\alpha^3 + \alpha^2$
α^{11}	$\alpha^3 + \alpha^2 + 1$
α^{12}	$\alpha + 1$
α^{13}	$\alpha^2 + \alpha$
α^{14}	$\alpha^3 + \alpha^2$
α^{15}	1

Donde $a^4 = a^3 + 1$

Este campo GF(2^4)es de *característica* 2, de *dimensión* 4, de *orden* 2^4=16 y tiene $2^4 - 1 = 15$ elementos

8

El exponenciador para GF(2^m)

- Se propone una implementación en hardware del ***método binario*** de exponenciación para GF(2^m) en su versión *LSB-first*.
- Esto nos ha permitido el diseño de una estructura que usa dos multiplicadores en paralelo mejorando la eficiencia del exponenciador.

9

Algoritmo de exponenciación

- Sea ***M*** un elemento arbitrario de GF(2^m) expresado como: $M = \sum_{i=0}^{i=m-1} m_i a^i$
- y sea *e (1£ e £ 2^m-1)* un entero cuya representación binaria es:

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

10

Algoritmo de exponenciación

- Entonces la potencia $R = M^e$ *modulo el polinomio irreducible G,* está también en $GF(2^m)$ y, según el método binario, se calcula mediante el siguiente algoritmo:

11

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-1* **do**
- 2.*a).-* **if** $e_i := 1$ **then** *R:=R*C (mod G)*
- 2.*b).-* *C:= C*C (mod G)*
- **end for** ;
- 3.- **return** *R*;

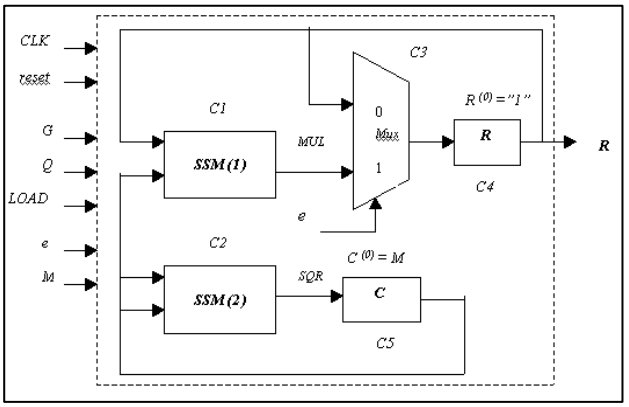
12

Ejemplo: e = 11111010 = 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	$M^{122} * M^{128} = M^{250}$	$(M^{128})^2 = M^{256}$

13

Arquitectura del exponenciador



Se requieren n multiplicaciones y sn^2 ciclos de reloj

14

Descripción VHDL

```
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;

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

15

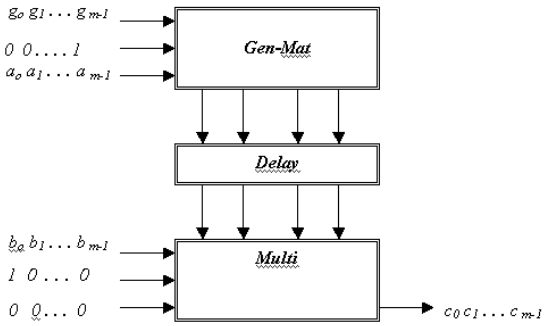
Resultados

	[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	$m^2-m+m/2+1$	$2m^2+2$	$(3m-1)n^2$

Requerimientos de hardware y de tiempos para la exponenciación

16

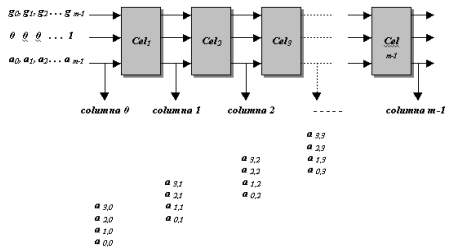
El multiplicador sistólico y serial



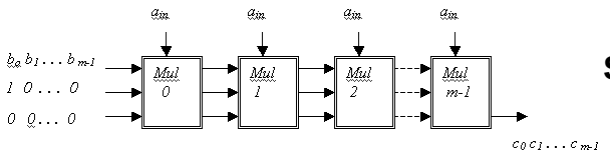
Hace una multiplicación en 3*m*-1 ciclos de reloj

17

Bloques del multiplicador



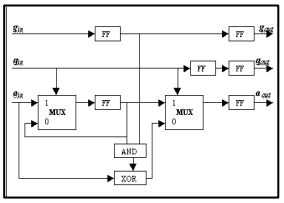
Gen_mat



Sol_multi

18

Celda celda01b



Celda de Gen_mat

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity celda01b is
5      port (CLK,CLR : in STD_LOGIC;
6            Gin,Qin,Ain: in STD_LOGIC;
7            Gout,Qout,Aout: out STD_LOGIC);
8  end celda01b;
9
10 architecture celda01b_arch of celda01b is
11
12 begin
13     process (CLK,CLR)
14     variable Gtemp, Qtemp, r: STD_LOGIC;
15     begin
16         if rising_edge(CLK) then
17           if (CLR='1')  then
18              Gout <='0';Qout<='0';Aout<='0';
19           elsif (Qin='1') then
20              Aout<=r; r:=Ain;
21           else
22              Aout<=((Gtemp and r) xor Ain);
23           end if;
24           Gout<=Gtemp; Gtemp:=Gin;
25           Qout<=Qtemp; Qtemp:=Qin;
26         end if;
27     end process;
28 end celda01b_arch;
```

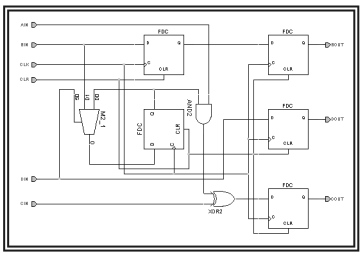
19

Descripción VHDL de Gen_mat

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity gen_mat4 is
5      generic (m: positive:=4);
6      port(CLK,CLR,Ginput,Qinput,Ainput:in STD_LOGIC;
7           Col_0,Col_1,Col_2,Col_3:out STD_LOGIC);
8  end gen_mat4;
9
10 architecture gen_mat4_arch of gen_mat4 is
11
12    component celda01b
13       port (CLK,CLR,Gin,Qin,Ain: in STD_LOGIC;
14             Gout,Qout,Aout: out STD_LOGIC);
15    end component;
16
17    signal Gtemp,Qtemp,Atemp: STD_LOGIC_VECTOR(0 to m-1);
18
19 begin
20
21    gen_mat: for I in 0 to m-1 generate
22
23        cel0: if (I=0) generate
24        celda0: celda01b port map (CLK=>CLK, CLR=>CLR, Gin=>Ginput, Qin=>Qinput,
25                        Ain=>Ainput,Gout=>Gtemp(I), Qout=>Qtemp(I), Aout=>Atemp(I));
26        end generate cel0;
27
28        cel1: if (I<=m-1 and I>=1) generate
29        celda1: celda01b port map (CLK=>CLK, CLR=>CLR, Gin=>Gtemp(I-1), Qin=>Qtemp(I-1)
30                        Ain=>Atemp(I-1),Gout=>Gtemp(I), Qout=>Qtemp(I), Aout=>Atemp(I)
31        end generate cel1;
32
33    end generate gen_mat;
34
35    Col_0<=Ainput; Col_1<=Atemp(0); Col_2<=Atemp(1); Col_3<=Atemp(2);
36
37 end gen_mat4_arch;
```

20

Celda cel_mul



```
library IEEE;
use IEEE.std_logic_1164.all;

entity cel_mul is
    port (CLK,CLR:in STD_LOGIC;
          Ain,Bin,Cin,Din: in STD_LOGIC;
          Bout,Cout,Dout: out STD_LOGIC);
end cel_mul;

architecture cel_mul_arch of cel_mul is
begin
   process (CLK,CLR)
   variable Btemp,r: STD_LOGIC;
   begin
      if rising_edge(CLK) then
         if (CLR='1') then
            Bout<='0';Dout<='0';Cout<='0';
         elsif Din='1' then
                  r:=Bin;
         end if;
         Dout<=Din;Cout<=(Cin xor (r and Ain));
         Bout<=Btemp;Btemp:=Bin;
      end if;
   end process;
end cel_mul_arch;
```


Descripción VHDL de Sol_multi

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Sol_multi is
    generic (m:positive:=4);
    port (CLK,CLR,Binput,Cinput,Dinput: in STD_LOGIC;
          Ainput: in STD_LOGIC_VECTOR(0 to m-1);
          Coutput: out STD_LOGIC);
end Sol_multi;

architecture Sol_multi_arch of Sol_multi is

   component cel_mul
      port (CLK,CLR,Ain,Bin,Cin,Din: in STD_LOGIC;
            Bout,Cout,Dout: out STD_LOGIC);
   end component;

   signal Btemp,Ctemp,Dtemp: STD_LOGIC_VECTOR(0 to m-1);

begin

   Sol: for I in 0 to m-1 generate

        cel0: if (I=0) generate
        celda0: cel_mul port map (CLK=>CLK, CLR=>CLR, Ain=>Ainput(I),
                        Bin=>Binput, Cin=>Cinput, Din=>Dinput,
                        Bout=>Btemp(I),Cout=>Ctemp(I),Dout=>Dtemp(I));
        end generate cel0;

        celx: if (I<=m-1 and I>=1) generate
        celdax: cel_mul port map (CLK=>CLK, CLR=>CLR, Ain=>Ainput(I),
                        Bin=>Btemp(I-1),Cin=>Ctemp(I-1),Din=>Dtemp(I-1),
                        Bout=>Btemp(I),Cout=>Ctemp(I),Dout=>Dtemp(I));
        end generate celx;

   end generate Sol;
   Coutput<=Ctemp(m-1);

end Sol_multi_arch;
```

Descripción VHDL del multiplicador(a)

```
1  library IEEE;
2  use IEEE.std_logic_1164.all;
3
4  entity Multiplicador is
5      generic (m: positive :=4);
6      port (CLK, CLR: in STD_LOGIC;
7            Gin,Qin,Ain,Cin,Bin: in STD_LOGIC;
8            Cout: out STD_LOGIC);
9  end Multiplicador;
10
11 architecture Multiplicador_arch of Multiplicador is
12
13   component gen_matx
14     port(CLK,CLR,Ginput,Qinput,Ainput:in STD_LOGIC;
15          Col_0:buffer STD_LOGIC;
16          Col:out STD_LOGIC_VECTOR(1 to m-1));
17   end component;
18
19   component delay1
20     port(CLK,CLR,In_Col_0: in STD_LOGIC;
21          In_Col: in STD_LOGIC_VECTOR (1 to m-1);
22          Out_Col_0: out STD_LOGIC;
23          Out_Col: out STD_LOGIC_VECTOR (1 to m-1));
24   end component;
25
26   component Sol_multi
27     port (CLK,CLR,Binput,Cinput,Dinput,A_0: in STD_LOGIC;
28           Ainput: in STD_LOGIC_VECTOR(1 to m-1);
29           Coutput: out STD_LOGIC);
30   end compoonent;
31
32   signal primera, segunda: STD_LOGIC_VECTOR(1 to m-1);
33   signal temp1,temp2: STD_LOGIC;
```

23

Descripción VHDL del multiplicador(b)

```
34
35 begin
36
37     U1:  gen_matx
38          port map(CLK=>CLK,CLR=>CLR,Ginput=>Gin,Qinput=>Qin,
39                   Ainput=>Ain, Col_0=>temp1,Col=>primera);
40     U2:  delay1
41          port map(CLK=>CLK,CLR=>CLR,In_Col_0=>temp1,
42                   In_Col=>primera,Out_Col_0=>temp2,Out_Col=>segunda);
43     U3:  Sol_multi
44          port map(CLK=>CLK,CLR=>CLR,Binput=>Bin,Cinput=>Cin,Dinput=>Qin,
45                   A_0=>temp2,Ainput=>segunda, Coutput=>Cout);
46                   Ain=>segunda, Bout=>Bout);
47 end Multiplicador_arch;
```

24

Requerimientos de hardware

	m=4	m=8	m=16	m=32	m=64	m=128
Reg=2.5m^2+11.5m-6	80	246	818	2922	10970	42426
AND=4m^2+12m-5	107	347	1211	4475	17147	67067
OR=1.5m^2+7.5m-2	52	154	502	1774	6622	25534
XOR=0.5m^2+1.5m-1	13	19	151	559	2141	8383

	FPGA XC4010XL	FPGA XCV300
No. de compuertas	10 K	323 K
CLB´s	400	1536
Flip-flops	1120	6144

25

Observaciones y consideraciones

- Siendo que el multiplicador es la parte modular del exponenciador, concluímos que si logramos una mejora en las complejidades en espacio y tiempo del multiplicador, lo podremos hacer directamente sobre el exponenciador.

26

Observaciones y consideraciones

- Examinamos una nueva arquitectura serial que se basa en el algoritmo *LSB-first* como se muestra enseguida.

27

Multiplicador serial LSB-first

- Sean $a, b, c \in GF(2^m)$ y se representan en una base polinomial como:
- $a = a_0 + a_1 a + \ldots + a_{m-1} a^{m-1}$
- $b = b_0 + b_1 a + \ldots + b_{m-1} a^{m-1}$
- $c = c_0 + c_1 a + \ldots + c_{m-1} a^{m-1}$

- La multiplicación c = a * b se puede expresar como:
- $c = a * b = (a_0 + a_1 a + \ldots + a_{m-1} a^{m-1}) * b$
- $c = (\ldots(((a_0 b) + a_1 b a) + a_2 b a^2) + \ldots) + a_{m-1} b a^{m-1}$

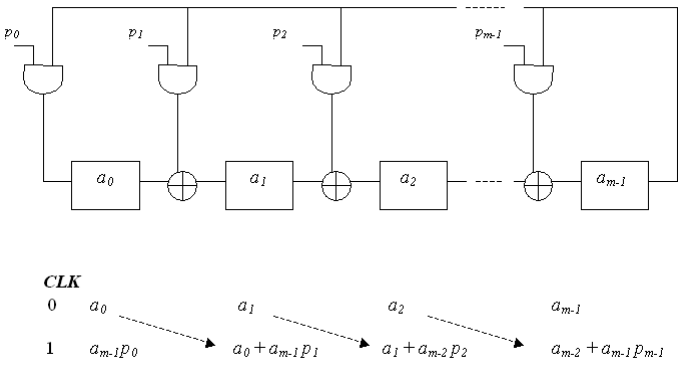
28

Multiplicación por *x*

- $D = xA(x) = a_0 + a_1 a + \ldots + a_{m-1} a^{m-1})$
- $D = a_0 x + a_1 x^2 + \ldots + a_{m-1} x^m$ ..... (1)
- $P(x) = p_0 + p_1 x + \ldots + p_{m-1} x^{m-1} + x^m$
- $x^m = p_0 + p_1 x + \ldots + p_{m-1} x^{m-1} \mod P(x)$ ....(2)
- **sust. (2) en (1):**
- $D = a_0 x + a_1 x^2 + \ldots + a_{m-1} [p_0 + p_1 x + \ldots + p_{m-1} x^{m-1}]$
- $D = [a_{m-1} p_0] + [a_0 + a_{m-1} p_1] x + \ldots + [a_{m-2} + a_{m-1} p_{m-1}] x^{m-1}$
- $D = d_0 + d_{1x} + \ldots + d_{m-1} x^{m-1}$
- *En general:*
- $d_0 = a_{m-1} p_0$ y $di = a_{i-1} + a_{m-1} p_i$

29

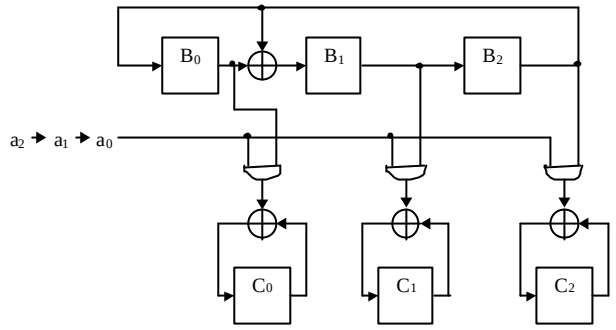
Arquitectura para *xA(x):*



$p_i \in$ GF(2) = {0,1} y se usa un ***L**inear **F**eedback **S**hift **R**egister*

30

Multiplicador LSB-first para GF(2^3) con $p(x)= x^3 +x +1$



Realiza una multiplicación en *m* ciclos de reloj.

31

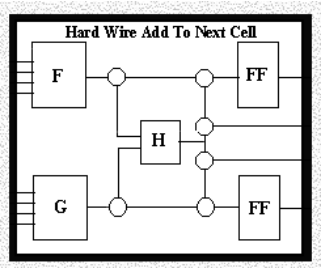
Observaciones y consideraciones

- Si realizamos el código VHDL y hacemos la síntesis como hasta hoy lo hemos hecho lo más probable es que sigamos enfrentando los mismos problemas en su implementación en el FPGA.
- Pero, examinando la estructura del FPGA, derivamos las siguientes observaciones:

32

Observaciones y consideraciones

- En los FPGA´s Virtex, cada slice cuenta con dos generadores de función de cuatro entradas.



33

Observaciones y consideraciones

- Cada generador puede ser utilizado para realizar funciones muy simples o más complejas.
- Ya sea una compuerta de dos entradas o un registro de corrimiento de 16 bits.
- Ojo!
  - Con 12 registros de corrimiento de 16 bits se puede construir un LFSR de 192 bits.

34

Observaciones y consideraciones

- Esto requiere de 12 generadores de función para construirlo, es decir de 6 slices del FPGA!
- Para un multiplicador de 192 bits, se han de agregar 192 celdas que ocuparán 1 flip-flop + 1 AND, requiriendo cada celda de 1/2 slice , es decir 96 slices para un total de 102 slices = 51 CLB´s !!

35

Trabajo inmediato

- Construcción del multiplicador SLB-first.
- Síntesis y simulación para verificación de resultados esperados.
- Usar este multiplicador como base para el diseño de un multiplicador por dígitos que mejorará considerablemente la complejidad en tiempo del corcuito.

36