UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Arquitecturas Secuenciales

Procesado Digital de la Señal en FPGA

Máster Universitario en Ingeniería de Sistemas Electrónicos

Objetivos

- Diseñar e implementar algoritmos utilizando arquitecturas secuenciales
- Diseñar e implementar algoritmos utilizando arquitecturas semi-paralelas
- Utilizar de manera eficiente los recursos de almacenamiento que nos ofrecen las FPGAs de Xilinx y Altera

Contenidos

- Arquitecturas secuenciales
- Arquitecturas semi-paralelas
- Recursos de almacenamiento

Arquitecturas secuenciales

⇒Frecuencia de muestreo(fs) < frecuencia de reloj (fclk)

⇒Si fs=fclk/c (Ts=c·Tclk), hay c ciclos de reloj para realizar todas las operaciones del algoritmo

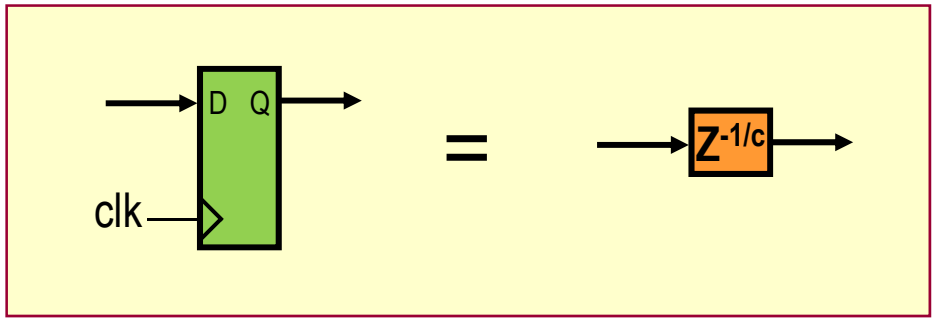
⇒La arquitectura necesita menos recursos que operaciones tiene el algoritmo

⇒Necesitamos los recursos suficientes para implementar el algoritmo en c ciclos

⇒Ej. Un algoritmo que tiene que hacer M=100 multiplicaciones siendo fs=fclk/c, con c=10 podrá implementarse con M/c=10 multiplicadores

⇒Se genera un Nuevo resultado cada c ciclos de reloj

⇒Cada registro implementa un retardo fracional de valor Ts/c



Arquitectura secuencial: Modelo Matlab de un filtro FIR

Filtro FIR M etapas:

$$y(n) = \sum_{k=0}^{M-1} h_k x(n-k)$$

Se calcula de forma iterativa en c ciclos

Matlab :

```
acc(i)=0;
for j=1:(M)
        acc(i)=acc(i)+x(i-j+1)*h(j);
end
y(i)=acc(i);
```

En cada iteración:

- Se multiplica un coef. por un dato
- Se acumula el resultado

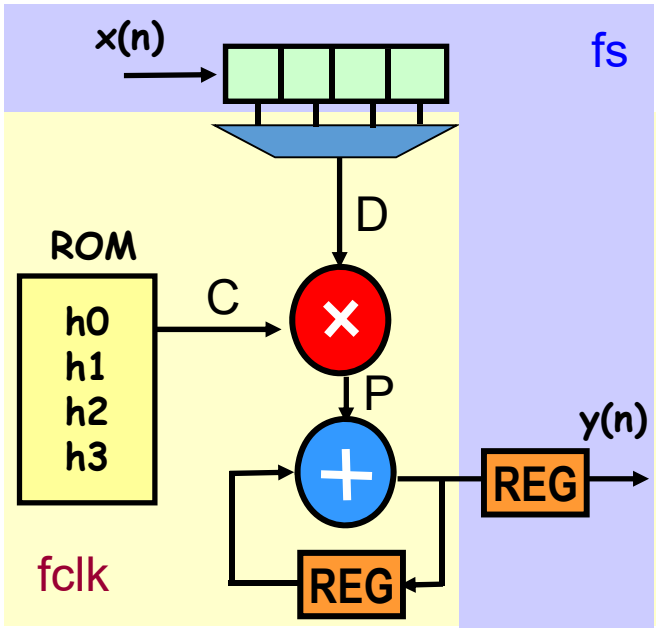
Requisitos Hardware:

- 1 Multiplicador
- 1 acumulador
- Recursos para almacenar al menos M muestras de entrada
- Recursos para almacenar los M coeficientes
- Lógica de control

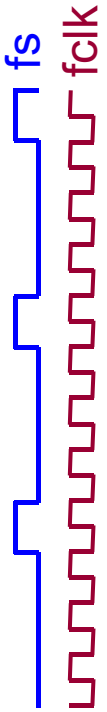
Arquitectura secuencial: Filtro FIR

Filtro FIR de 4 etapas:

$$y(n) = \sum_{k=0}^{3} h_k x(n-k) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3)$$



$$T_{max} = f_{clk}/M = f_{clk}/4$$



cycle	D	C	P	Y(n)
0	X(0)	h0	x(0) h0	x(0) h0
1	0	h1	0	x(0) h0
2	0	h2	0	x(0) h0
3	0	h3	0	x(0) h0
4	X(1)	h0	x(1) h0	x(1) h0
5	X(0)	h1	x(0) h1	x(1) h0+x(0) h1
6	0	h2	0	x(1) h0+x(0) h1
7	0	h3	0	x(1) h0+x(0) h1
8	X(2)	h0	x(2) h0	x(2) h0
9	X(1)	h1	x(1) h1	x(2) h0+x(1) h1
10	X(0)	h2	x(0) h2	x(2) h0+x(1) h1+x(0) h3
11	0	h3	0	x(2) h0+x(1) h1+x(0) h3

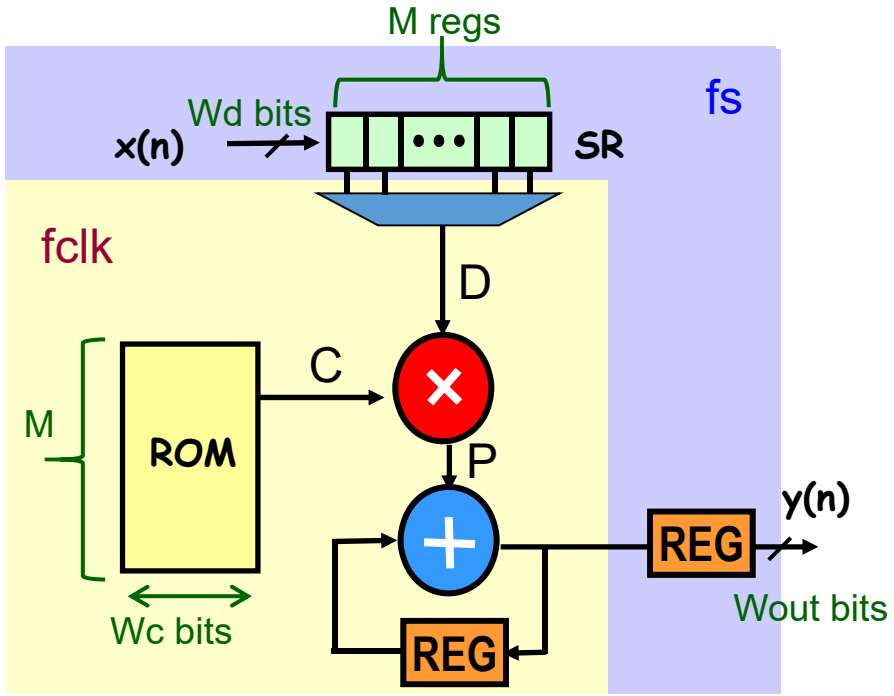
- MUX, ROM, MAC activos cada ciclo
- SR activo cada 4 ciclos

Cycles	Shift-register (SR)			
0	x(0)	0	0	0
4	x(1)	x(0)	0	0
8	x(2)	x(1)	x(0)	0
12	x(3)	x(2)	x(1)	x(0)
16	x(4)	x(3)	x(2)	x(1)

Arquitectura secuencial: Filtro FIR

Filtro FIR de M etapas:

$$y(n) = \sum_{k=0}^{M-1} h_k x(n-k)$$



Recursos HW:

- M registros de Wd bits
- ROM: MxWc bits
- Multiplicador: WdxWc bits
- Sumador: Wd+Wc+g bits
- Registro (acc.): Wd+Wc+g bits
  (g: bits de guarda del acumulador)
- Registro de salida: Wout bits

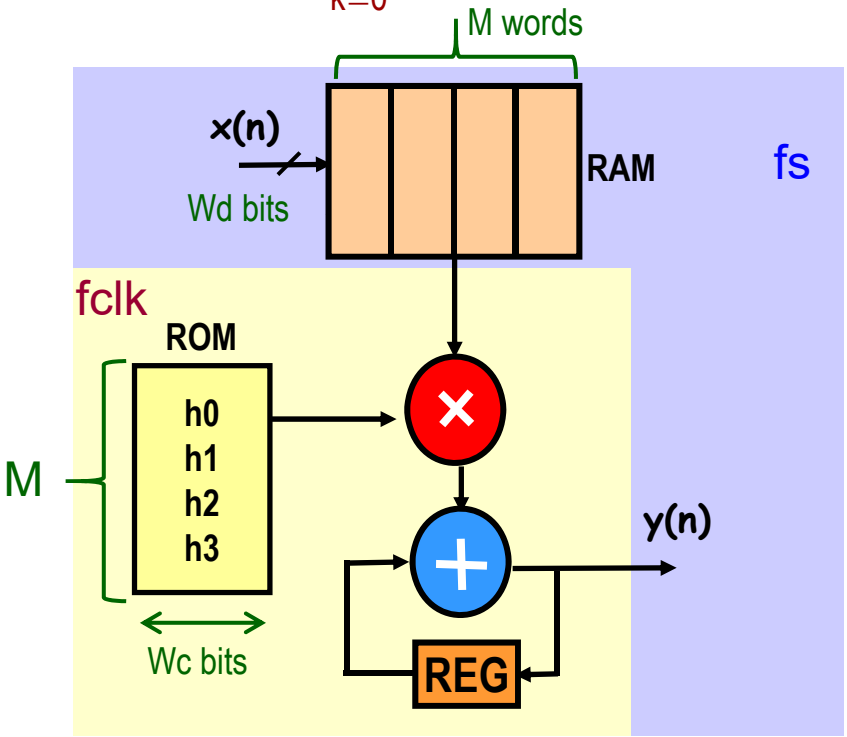
Throughput:

- T= fclk/Cp, with Cp≥M

$$T_{max} = f_{clk} / M$$

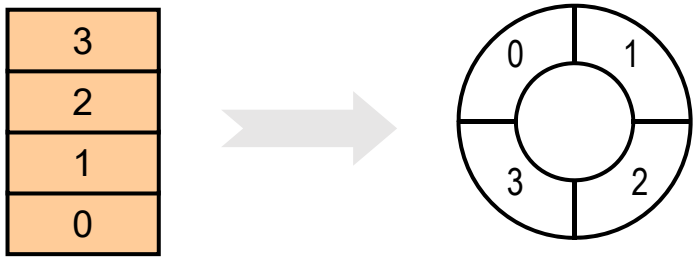
Arquitectura secuencial: Filtro FIR

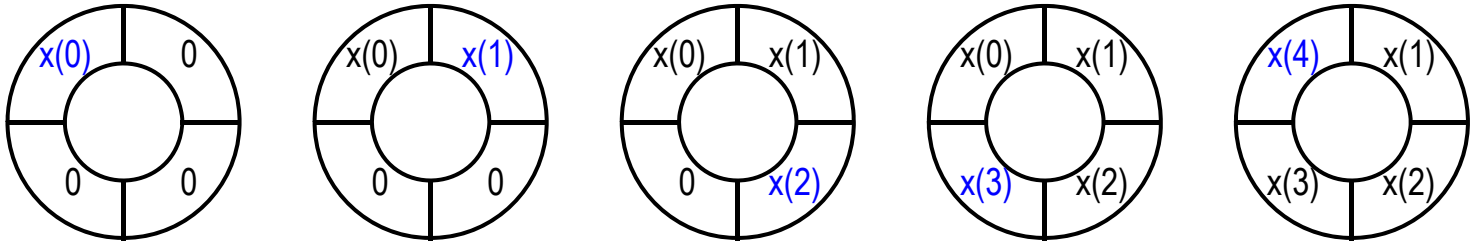
$$y(n) = \sum_{k=0}^{M-1} h_k x(n-k) = h_0 x(n) + h_1 x(n-1) + h_2 x(n-2) + h_3 x(n-3)$$



Para filtros de orden elevado es ineficiente el uso de shift-registers

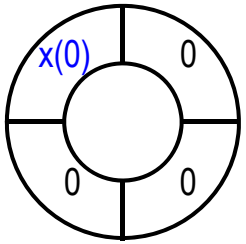
⟹ **Buffers circulares implementados en RAM MxWd bits**



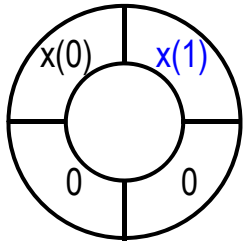


ciclo 0 ciclo 4 ciclo 8 ciclo 12 ciclo 16

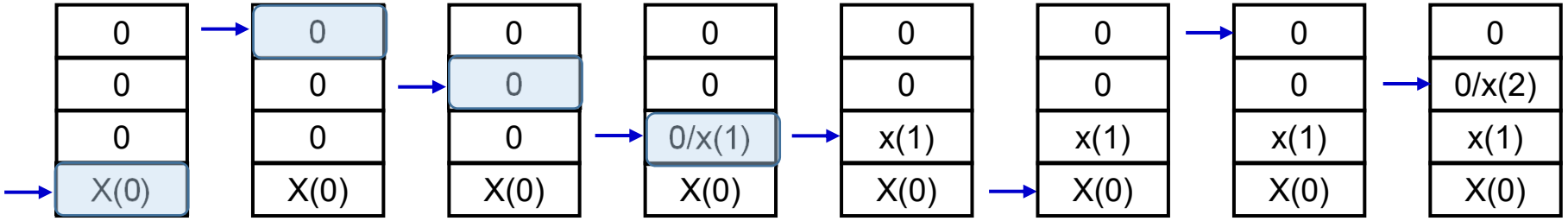
Buffer circular: FIR M=4



cycle 0

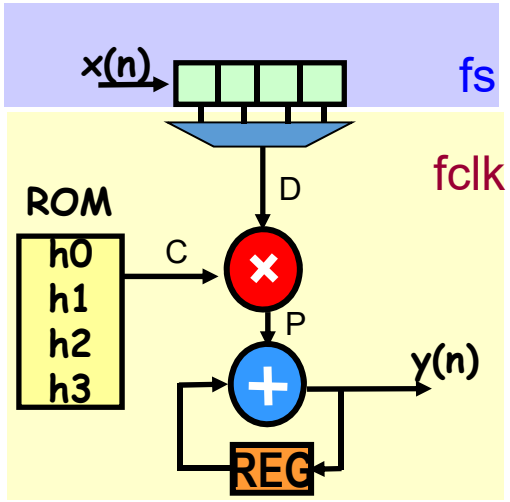


cycle 4

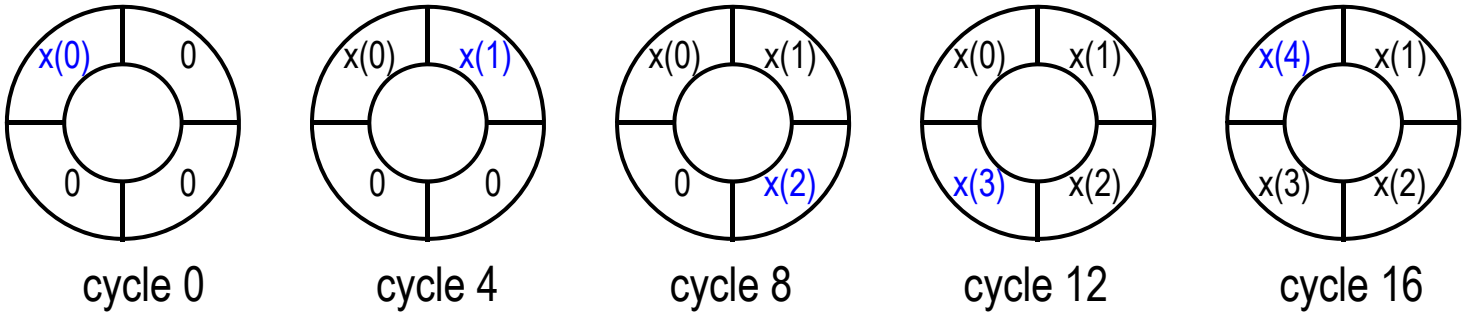


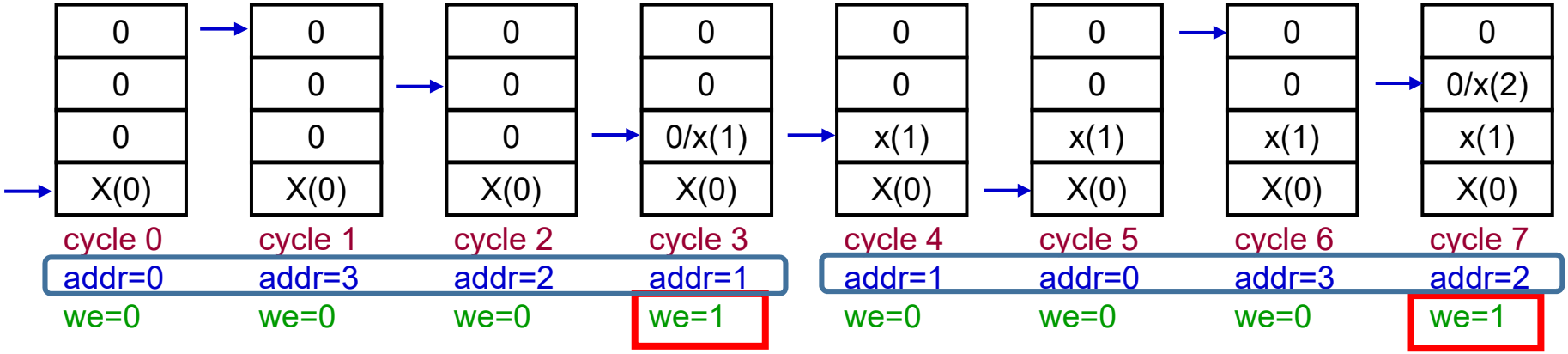
cycle 0 addr=0 we=0 | cycle 1 addr=3 we=0 | cycle 2 addr=2 we=0 | cycle 3 addr=1 **we=1** | cycle 4 addr=1 we=0 | cycle 5 addr=0 we=0 | cycle 6 addr=3 we=0 | cycle 7 addr=2 **we=1**

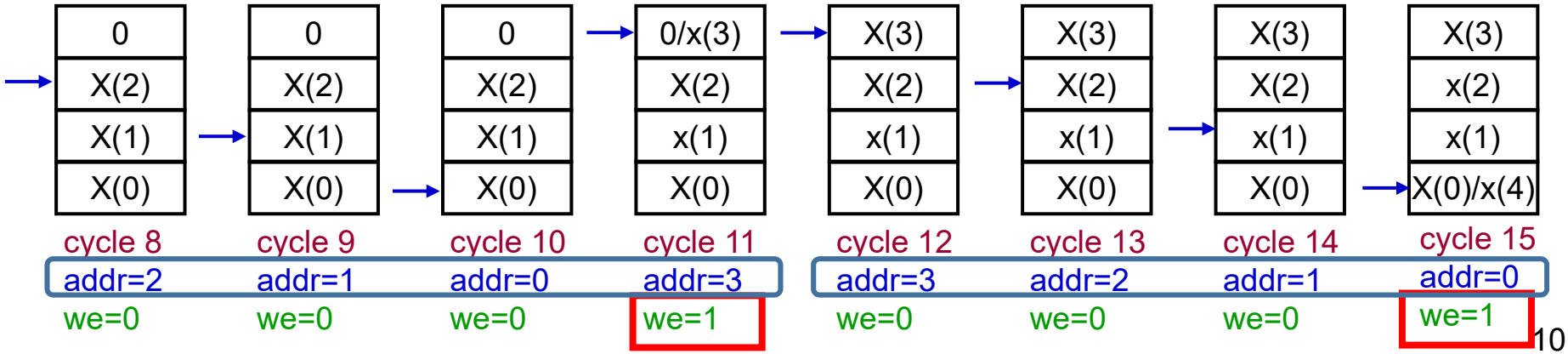
cycle	D	C	P	Y(n)
0	X(0)	h0	x(0) h0	x(0) h0
1	0	h1	0	x(0) h0
2	0	h2	0	x(0) h0
3	0	h3	0	x(0) h0
4	X(1)	h0	x(1) h0	x(1) h0
5	X(0)	h1	x(0) h1	x(1) h0+x(0) h1
6	0	h2	0	x(1) h0+x(0) h1
7	0	h3	0	x(1) h0+x(0) h1



Buffer circular: FIR M=4

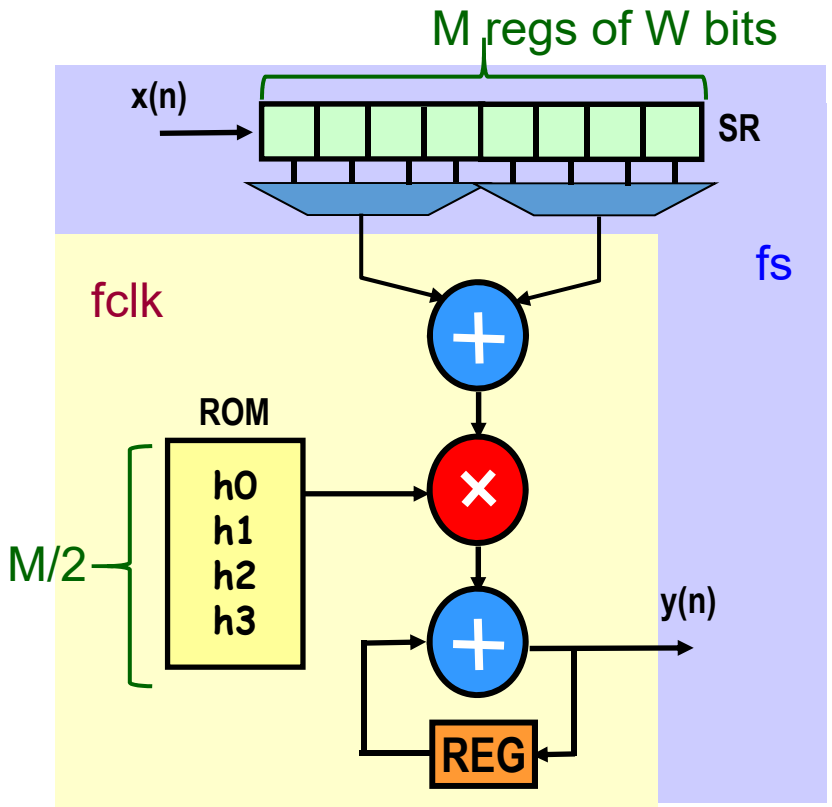






Aprovechando las simetrías

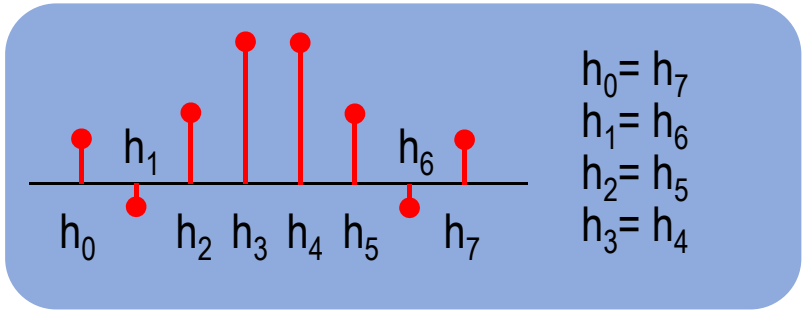
Filtro FIR de fase lineal con coeficientes simétricos



$$T_{max} = f_{clk}/(M/2) = 2 \cdot f_{clk}/M$$

FIR simétrico de 8 etapas

$$y(n)=h_0x(n)+h_1x(n-1)+h_2x(n-2)+h_3x(n-3)+h_4x(n-4)+h_5x(n-5)+h_6x(n-6)+h_7x(n-7)$$

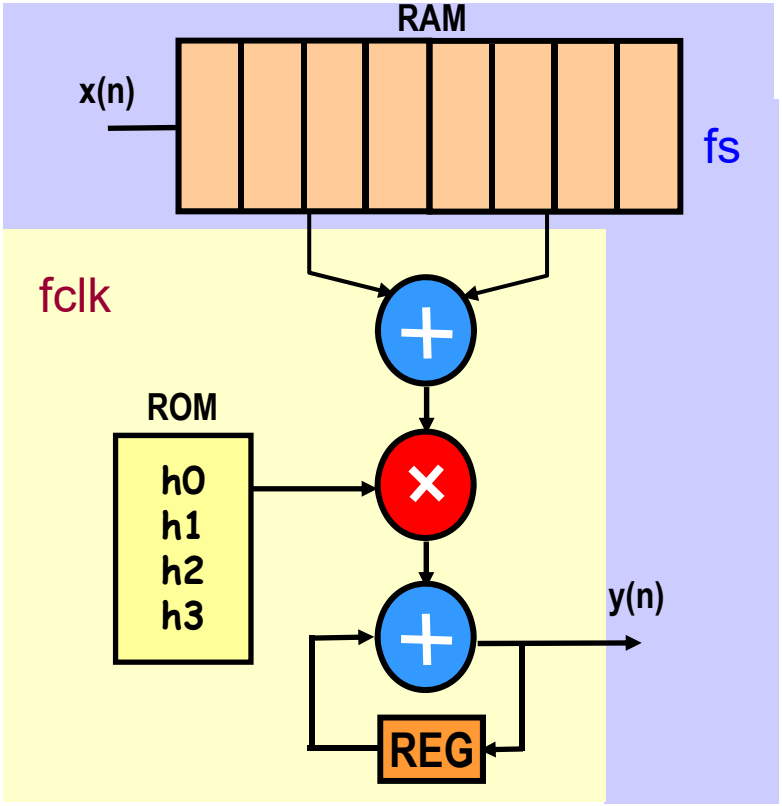


$$y(n)=h_0[x(n)+x(n-7)]+h_1[x(n-1)+x(n-6)]+h_2[x(n-2)+x(n-5)]+h_3[x(n-3)+x(n-4)]$$

- Necesita la mitad de multiplicadores
- Alcanza doble throughput

Arquitectura secuencial: FIR simétrico

Filtro FIR de fase lineal con coeficientes simétricos



Buffer circular implementado en RAM
Se necesitan dos accesos a memoria por cada ciclo de reloj

⇒ **Memoria RAM de doble puerto**

$$T_{max} = f_{clk} / (M/2) = 2 \cdot f_{clk} / M$$

Incrementando el paralelismo: Modelo Matlab de un filtro semi-paralelo

Filtro FIR de M etapas:

$$y(n) = \sum_{k=0}^{M-1} h_k x(n-k)$$

Ej: se calcula iterando cada M/4 ciclos

Matlab model:

```
h0=h(1:M/4);
h1=h(M/4+1:2*M/4);
h2=h(2*M/4+1:3*M/4);
h3=h(3*M/4+1:M);

acc(i)=0;
for j=1:(M/4)
    m0=xn(i-j+1)*h0(j);
    m1=xn(i-j-M/4+1)*h1(j);
    m2=xn(i-j-2*M/4+1)*h2(j);
    m3=xn(i-j-3*M/4+1)*h3(j);
    acc(i)=acc(i)+m0+m1+m2+m3;
end
y(i)=acc(i);
```

Ejecución en 1 ciclo de clk

En cada iteración:

- Se direccionan 4 coefs. y 4 datos de entrada
- 4 muestras se multiplicand por 4 coef.
- Los 4 resultados de la multiplicación se acumulan

Requisitos Hardware:

- 4 multiplicadores
- 3 sumadores y 1 acumulador
- Recursos para almacenar las 4 últimas muestras y acceso a ellas simultáneo
- Recurso para almacenar 4 coeficientes y acceso simultáneo
- Lógica de control

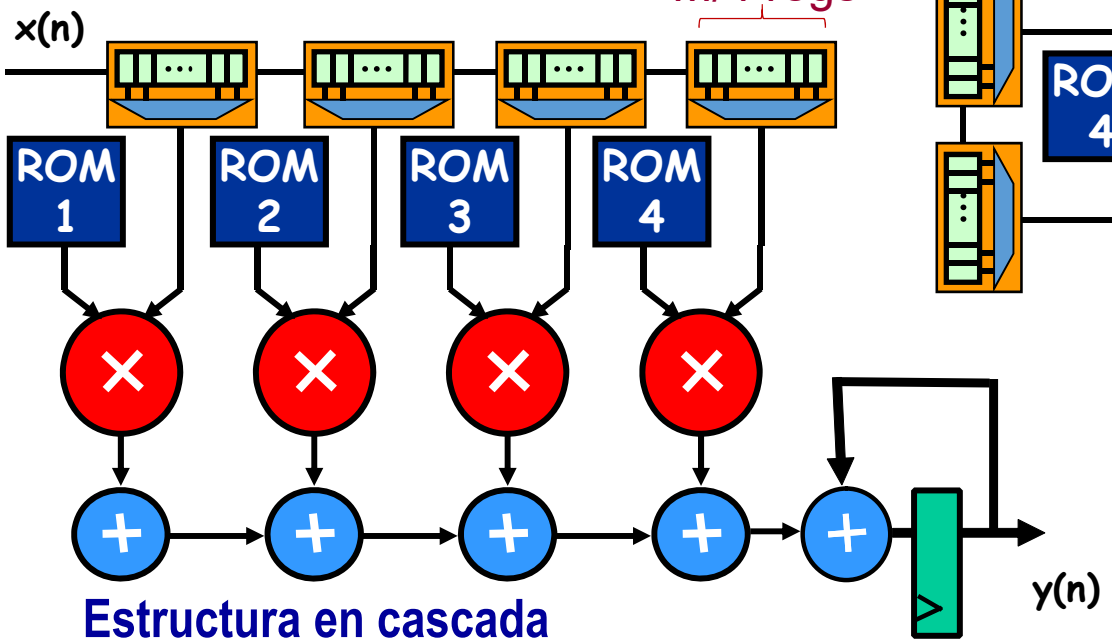
Arquitectura semiparalela: filtro FIR

¿throughput elevado?

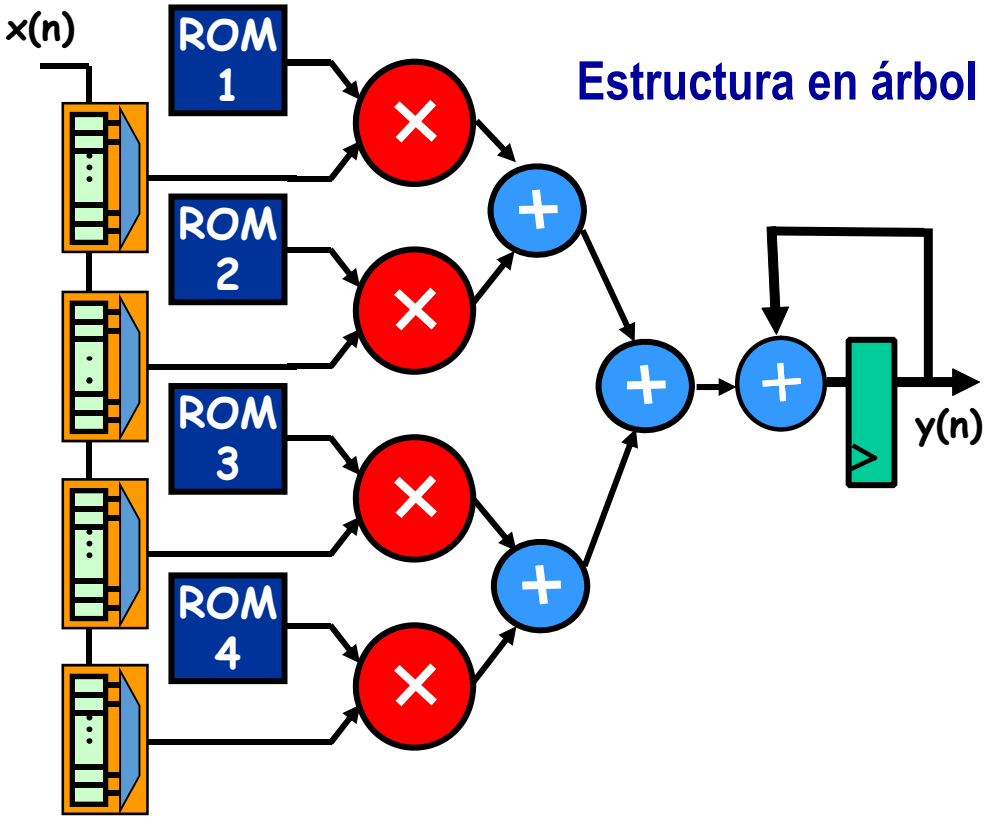
⇒Elevado paralelismo en la unidad MAC

⇒Ofrece un resultado cada M/4 ciclos de reloj

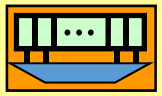
$$T_{\max} = {4 f_{clk}}/{M}$$



Estructura en cascada



Estructura en árbol

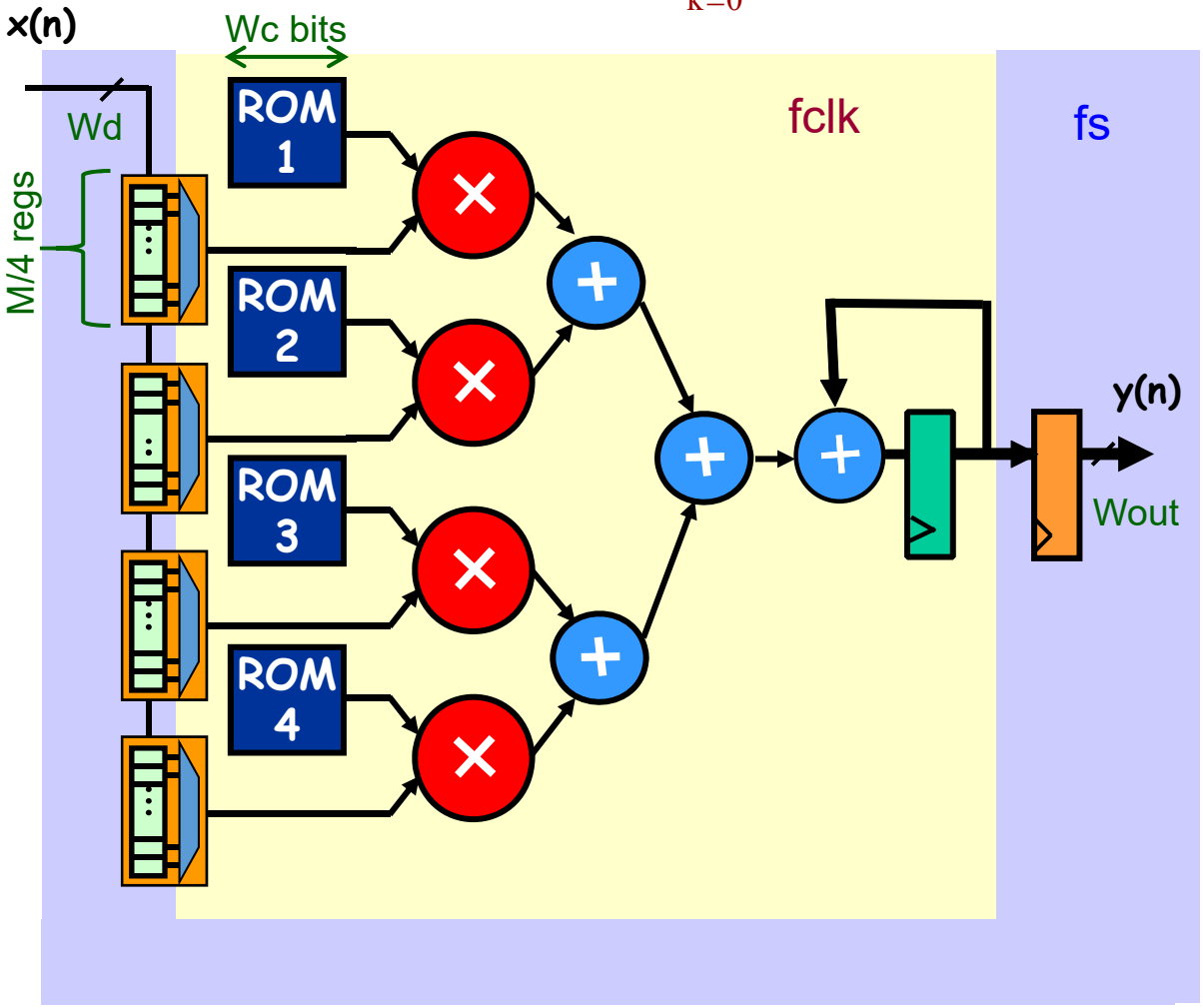


Se implementa con RAM

Arquitectura semiparalela: filtro FIR

Filtro FIR de M etapas:

$$y(n) = \sum_{k=0}^{M-1} h_k x(n-k)$$



Recursos HW:

- M registros of Wd bits
- 4 ROM: M/4xWc bits
- 4 Multiplicadores: WdxWc bits
- 4 sumadores: Wd+Wc+g bits
- Registros (acc.): Wd+Wc+g bits
  (g: bit de guarda del acumulador)
- Registro de salida: Wout bits

Throughput:

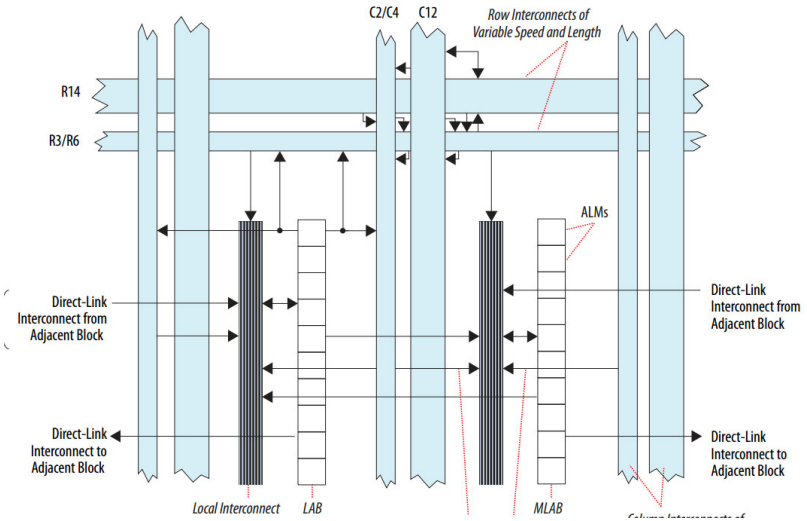
- T= fclk/Cp, con Cp≥M/4

$$T_{max} = 4 f_{clk} / M$$

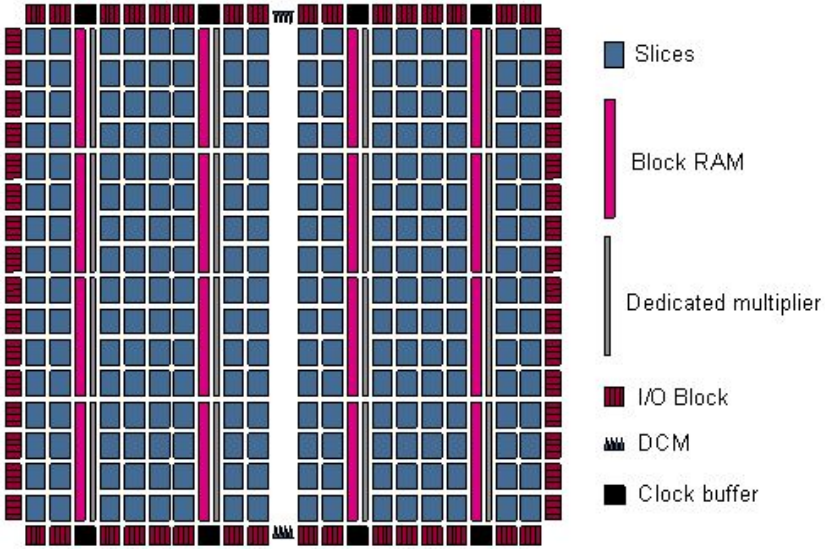
Recursos de almacenamiento

- Memoria Embebida: bloques de recursos dedicados específicamente a implementar memorias (M10k blocks en Altera, BRAM en Xilinx)
- Memoria Distribuida: se implementa configurando los bloques lógicos (MLAB en el caso de Altera, CLB Slices en el caso de Xilinx)

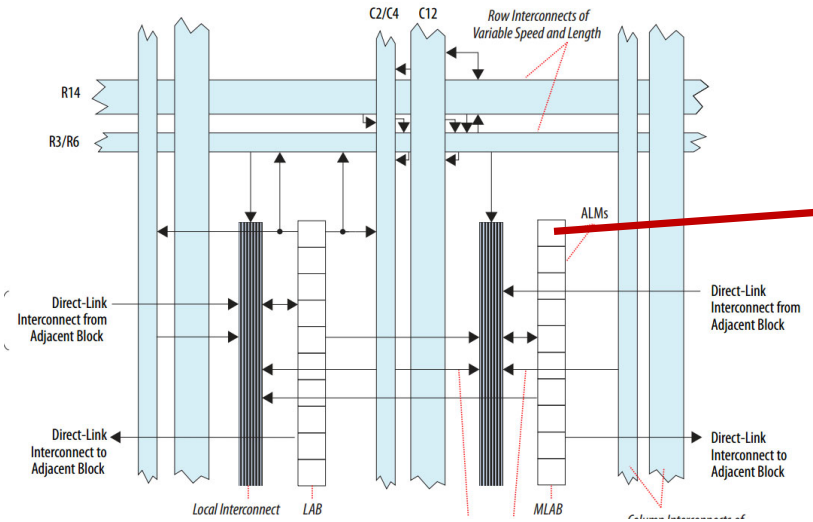
Altera Cyclone-V

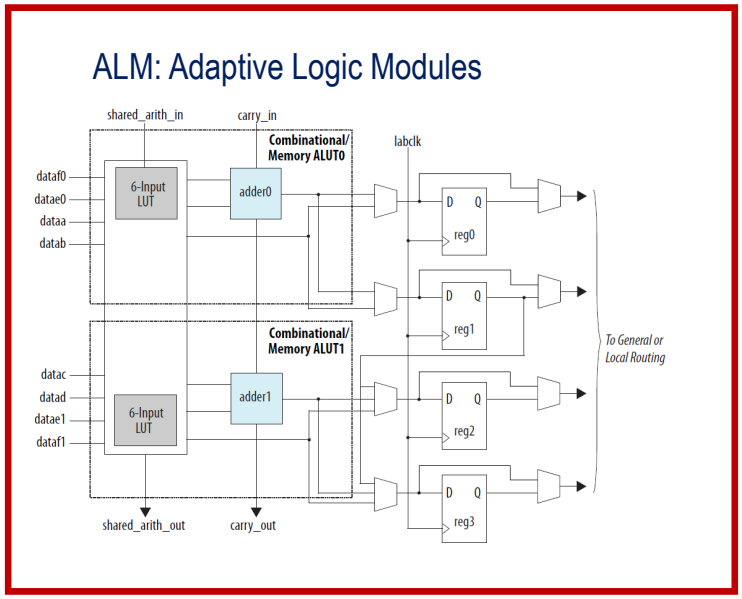


Xilinx Virtex- 7

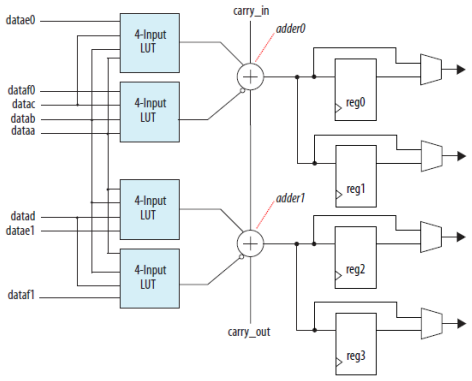


Memoria Distribuida en Cyclone-V (Altera)

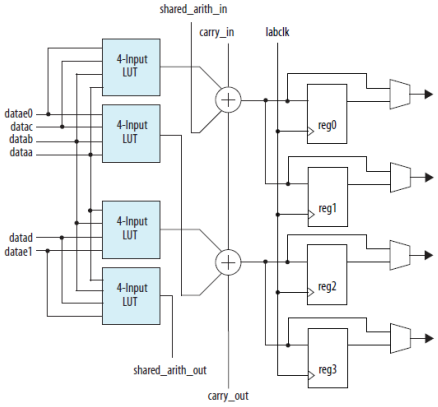




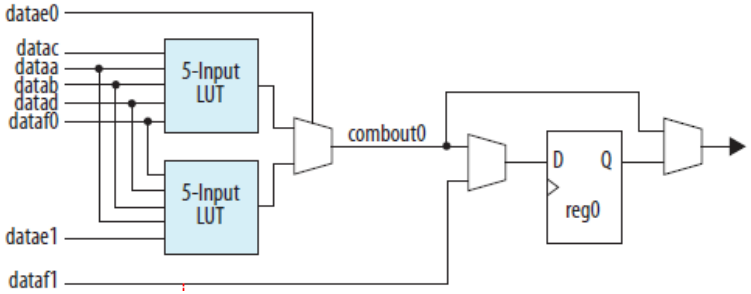
Arithmetic Mode



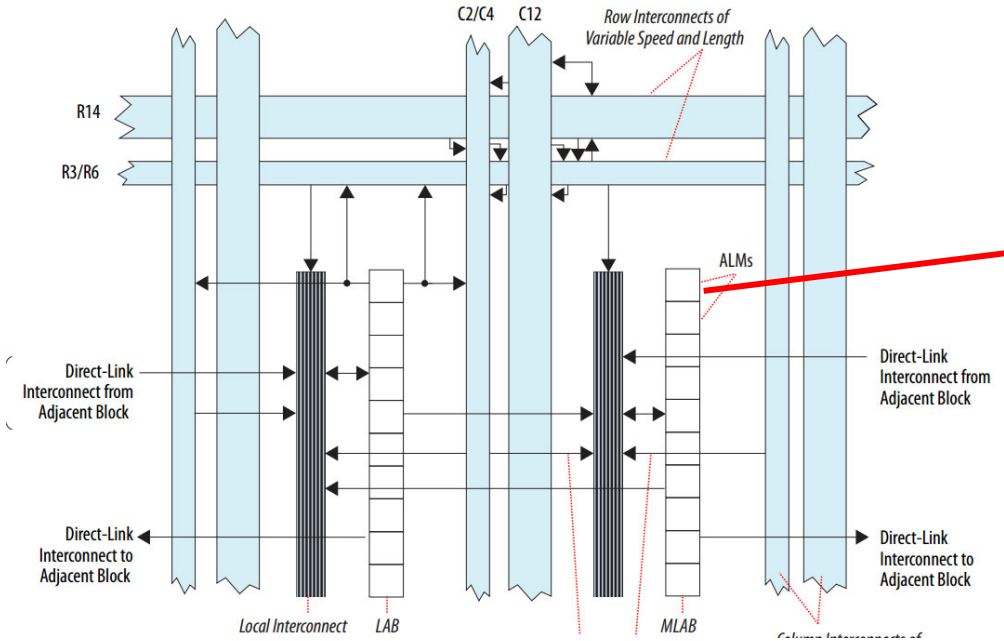
Shared Arithmetic Mode

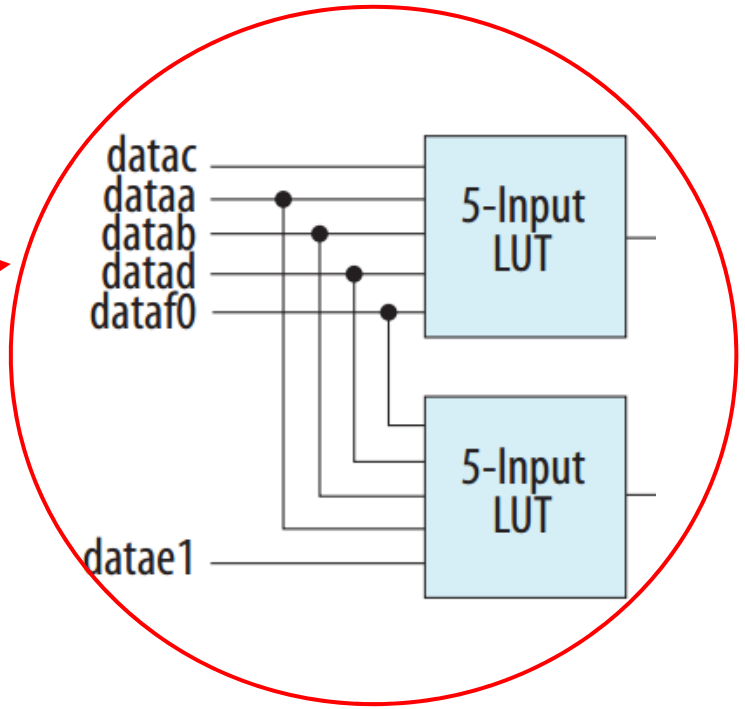


Extended LUT



Memoria Distribuida en Cyclone-V (Altera)



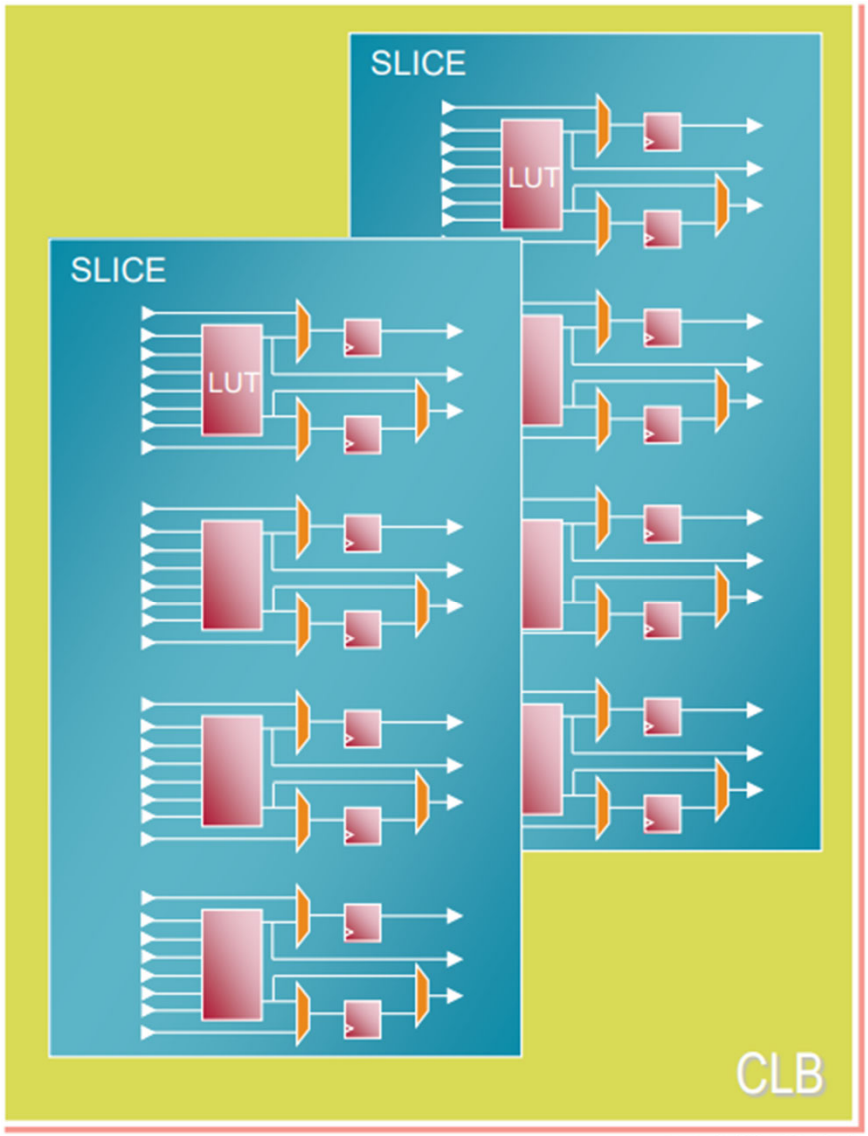


Cada MLAB equivale a una SRAM de doble puerto de 640 bits.

Cada ALM puede configurarse como una SRAM de doble puerto de 32x2 bits

Si 1 MLAB contiene 10 ALM → 32x20 SRAM DP

Memoria Distribuida en Virtex-7 (Xilinx)



- Dos SLICES en cada CLB
- Cada SLICE M de la Virtex-7 contiene 8 LUTs de 6 entradas
- Dos flip-flops por cada LUT

XILINX.

Memoria Distribuida en Virtex-7 (Xilinx)

Posibles configuraciones	Número de LUTs
32 x (1 - 16) bits (Puerto simple)	1 - 8
32 x (1 – 8) bits (Doble puerto)	2 – 8
32 x (1 – 4) bits (Cuádruple puerto)	4 - 8
32 x (1 – 14) bits (simple-doble puerto)	2 - 8
64 x (1 - 8) bits (Puerto simple)	1 - 8
64 x (1 – 4) bits (Doble puerto)	2 - 8
64 x (1 – 2) bits (Cuádruple puerto)	4 - 8
64 x 1 bits (Ocho puertos)	8
64 x 1 bit (Cuádruple puerto)	8
64 x (1 -7) bits (simple-doble puerto)	2 - 8
128 x (1 – 4) bits (Puerto simple)	2 - 8
128 x (1 – 2) bit (Doble puerto)	4 - 8
128 x 1 bit (Cuádruple puerto)	8
256 x 1 bit (Puerto simple)	8
256 x 1 bit (Doble puerto)	8
512 x 1 bit (Puerto simple)	8

Cada SLICE M de la Virtex-7 contiene 8 LUTs de 6 entradas.

Puerto simple

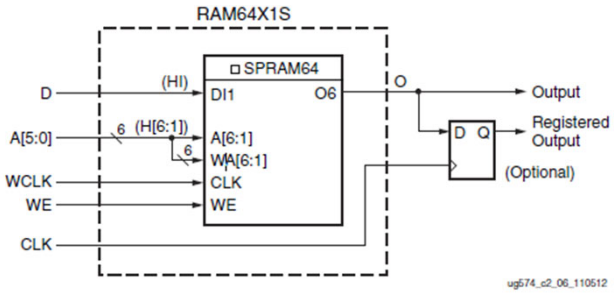


Figure 2-5: **Distributed RAM (RAM64X1S)**

Puerto doble

Lectura y escritura en la misma dirección

Lectura en diferentes direcciones

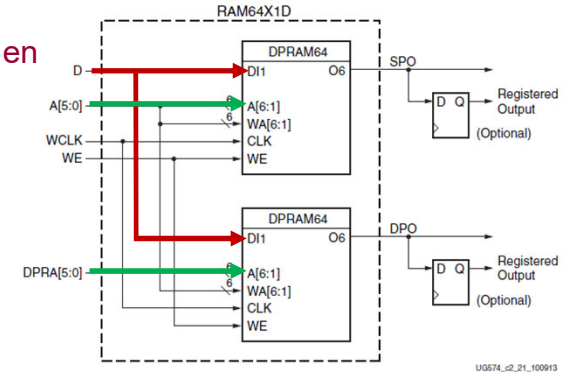


Figure 2-6: **64x1 Dual-Port Distributed RAM (RAM64X1D)**

Se puede extender a una de 64x1 de puerto cuádruple duplicando el número de LUTs o a una de ocho puertos utilizando 8 LUTs

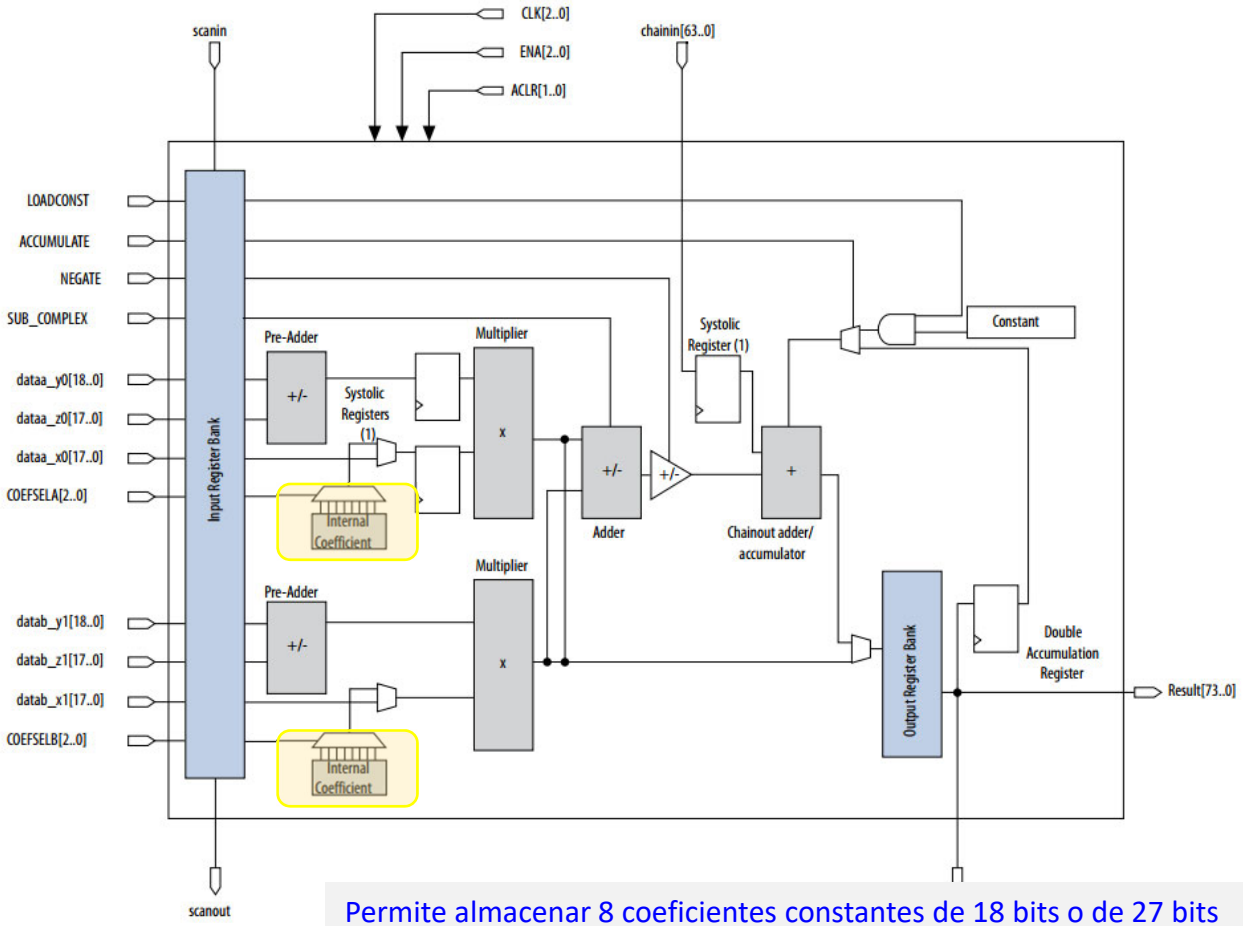
Memoria distribuida

Las memorias distribuidas son adecuadas para:

- Almacenar los coeficientes de un filtro cuando su número no es demasiado elevado
- Implementar registros de desplazamiento (shift-registers)
- Implementar FIFOs
- Implementar líneas de retardo

Memorias incluidas en bloques DSP

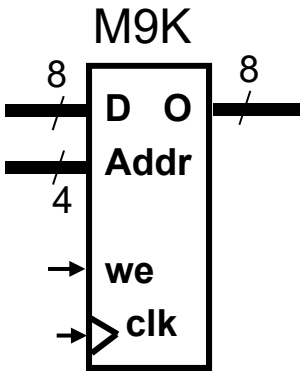
Altera _Cyclone V DSP



Permite almacenar 8 coeficientes constantes de 18 bits o de 27 bits

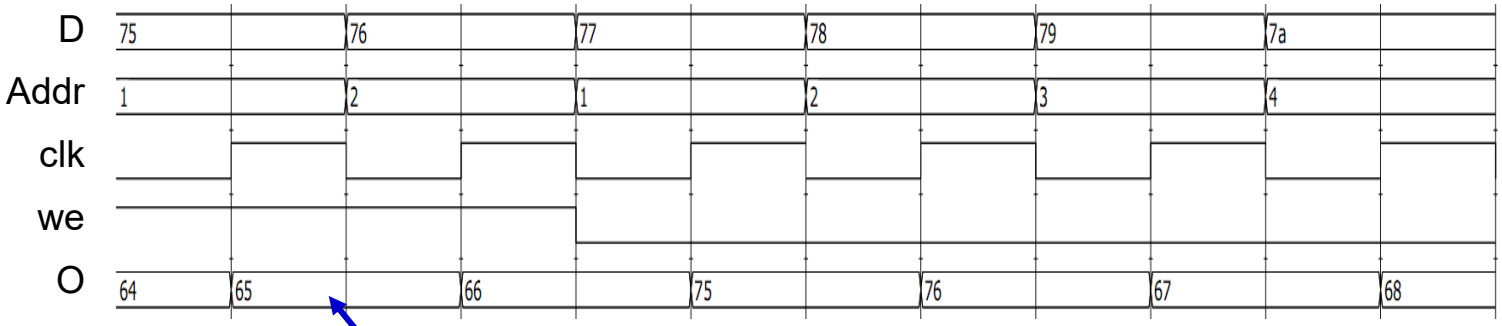
Read-first RAM en Verilog

- Lectura síncrona: read first

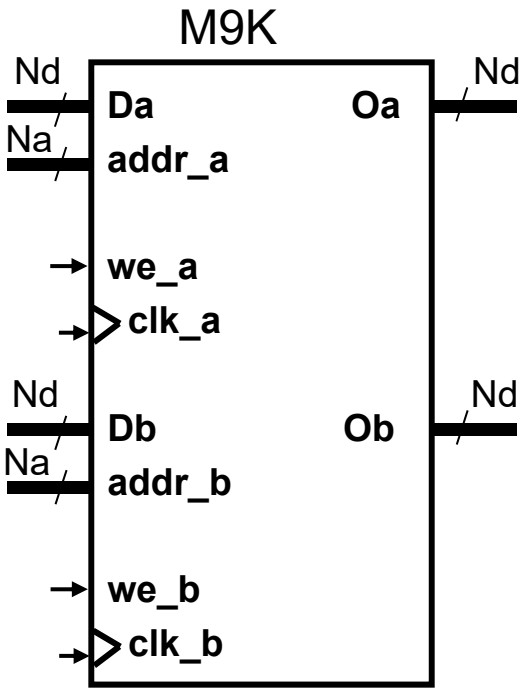



```
read [7:0] D;
input [3:0] Addr;
input clk,we;
output reg [7:0] O;

reg [7:0] RAM[15:0];

always@(posedge clk)
    begin
        if (we)
            RAM[Addr] <= D;
            O <= RAM[Addr];
    end
```

⇒ M9K como RAM 16x8-bit de puerto simple



RAM de doble Puerto con Verilog



⇒M9K como 2^NaxNd-bit RAM de doble puerto

```
parameter Nd=8; //DATA WIDTH
parameter Na=6); //ADDR WIDTH

wire [(Nd-1):0] Da, Db;
wire [(Na-1):0] addr_a, addr_b;
wire we_a, we_b, clk_a, clk_b;
reg [(Nd-1):0] Oa, Ob;

reg [Nd-1:0] ram[2**Na-1:0];

always @ (posedge clk_a)  // Port A
      if (we_a)
        begin
           ram[addr_a] <= Da;
           Oa <= Da;
        end
      else
         Oa <= ram[addr_a];

always @ (posedge clk_b) // Port B
      if (we_b)
         begin
             ram[addr_b] <= Db;
       Ob <= Db;
         end
      else
         Ob <= ram[addr_b];
```

ROM en Verilog HDL

```
wire [3:0] addr;
wire clk;
reg [7:0] q;

reg [7:0] rom[2**4-1:0];


 initial
    begin
       $readmemh("rom_init.txt", rom);
   end

 always @ (posedge clk)
      q <= rom[addr];
```

```
always @ (addr)
      q <= rom[addr];
```

rom_init.txt

Lectura del fichero de datos (en hexadecimal), si binario ($readmemb)

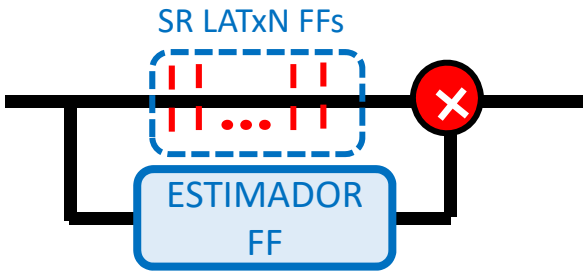
ROM con lectura síncrona (Bloques M9K)

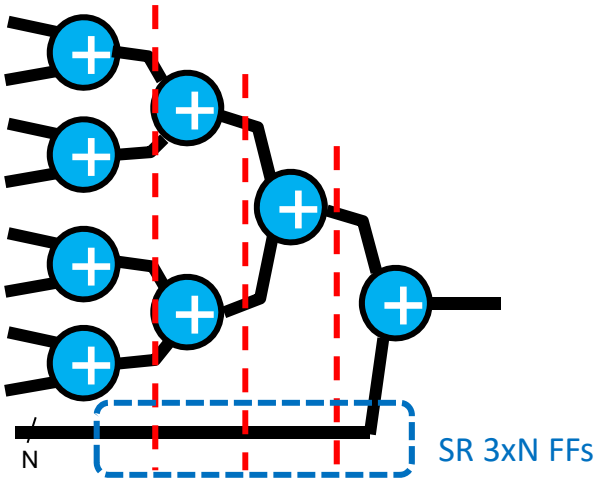
ROM con lectura asíncrona (LUTs)

```
0f
f3
2f
3a
d1
55
7f
7f
f2
92
af
bd
c1
d5
6f
ff
```

Registros de desplazamiento en DSP

- Operador necesario en la implementación HW de algoritmos DSP
  - Buffers
  - Segmentación
  - Sincronización de operadores
- FPGA actuales → además de FFs, incluyen recursos para implementación eficiente
  - Memorias embebidas
  - Uso de los registros de configuración de las LUTs





Registros de desplazamiento en Cyclone IV

```
wire clk,ce;
wire [7:0] D;
wire [7:0] Q;

reg [7:0] SR [15:0];

integer i;

always @ (posedge clk)
   if (ce)
      begin
         SR[0] <= D0;
         for (i = 15; i>0; i = i-1)
            SR[i] <= SR[i-1];
      end

assign Q = SR[3];
```

16x8 FFs =128 FFs

(* ramstyle = "logic" *) reg [7:0] SR [15:0];

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Total logic elements	128 / 6,272 (2 %)
1	-- Combinational with no register	0
2	-- Register only	128
3	-- Combinational with a register	0
2		
3	Logic element usage by number of LUT inputs	
1	-- 4 input functions	0
2	-- 3 input functions	0
3	-- <=2 input functions	0
4	-- Register only	128
4		
5	Logic elements by mode	
1	-- normal mode	0
2	-- arithmetic mode	0
6		
7	Total registers*	128 / 6,684 (2 %)
1	-- Dedicated logic registers	128 / 6,272 (2 %)
2	-- I/O registers	0 / 412 (0 %)
8		
9	Total LABs: partially or completely used	8 / 392 (2 %)
10	Virtual pins	0
11	I/O pins	18 / 92 (20 %)
1	-- Clock pins	1 / 3 (33 %)
2	-- Dedicated input pins	0 / 9 (0 %)
12		
13	M9Ks	0 / 30 (0 %)

(* ramstyle = “M9K" *) reg [7:0] SR [15:0];

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage
1	Total logic elements	18 / 6,272 (< 1 %)
1	-- Combinational with no register	6
2	-- Register only	8
3	-- Combinational with a register	4
2		
3	Logic element usage by number of LUT inputs	
1	-- 4 input functions	1
2	-- 3 input functions	4
3	-- <=2 input functions	5
4	-- Register only	8
4		
5	Logic elements by mode	
1	-- normal mode	6
2	-- arithmetic mode	4
6		
7	Total registers*	12 / 6,684 (< 1 %)
1	-- Dedicated logic registers	12 / 6,272 (< 1 %)
2	-- I/O registers	0 / 412 (0 %)
8		
9	Total LABs: partially or completely used	3 / 392 (< 1 %)
10	Virtual pins	0
11	I/O pins	18 / 92 (20 %)
1	-- Clock pins	1 / 3 (33 %)
2	-- Dedicated input pins	0 / 9 (0 %)
12		
13	M9Ks	1 / 30 (3 %)

Registros de desplazamiento en Cyclone V

(* ramstyle = "logic" *) reg [7:0] SR [15:0];

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	33 / 56,480	< 1 %
2	ALMs needed [=A-B+C]	33	
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	8 / 5,648	< 1 %
7			
8	Combinational ALUT usage for logic	1	
9	Combinational ALUT usage for route-throughs	78	
10			
11	Dedicated logic registers	128	
12			
13	Virtual pins	0	
14	I/O pins	18 / 268	7 %
15			
16	M10K blocks	0 / 686	0 %
17	Total MLAB memory bits	0	
18	Total block memory bits	0 / 7,024,640	0 %
19	Total block memory implementation bits	0 / 7,024,640	0 %
20			
21	Total DSP Blocks	0 / 156	0 %

(* ramstyle = “M10K" *) reg [7:0] SR [15:0];

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	11 / 56,480	< 1 %
2	ALMs needed [=A-B+C]	11	
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	3 / 5,648	< 1 %
7			
8	Combinational ALUT usage for logic	15	
9	Combinational ALUT usage for route-throughs	7	
10			
11	Dedicated logic registers	32	
12			
13	Virtual pins	0	
14	I/O pins	18 / 268	7 %
15			
16	M10K blocks	1 / 686	< 1 %
17	Total MLAB memory bits	0	
18	Total block memory bits	104 / 7,024,640	< 1 %
19	Total block memory implementation bits	10,240 / 7,024,640	< 1 %
20			
21	Total DSP Blocks	0 / 156	0 %

(* ramstyle = “MLAB" *) reg [7:0] SR [15:0];

Fitter Resource Usage Summary

<<Filter>>

	Resource	Usage	%
1	Logic utilization (ALMs needed / total ALMs on device)	17 / 56,480	< 1 %
2	ALMs needed [=A-B+C]	17	
3			
4	Difficulty packing design	Low	
5			
6	Total LABs: partially or completely used	4 / 5,648	< 1 %
7			
8	Combinational ALUT usage for logic	7	
9	Combinational ALUT usage for route-throughs	12	
10	Memory ALUT usage	8	
11			
12			
13	Dedicated logic registers	20	
14			
15	Virtual pins	0	
16	I/O pins	18 / 268	7 %
17			
18	M10K blocks	0 / 686	0 %
19	Total MLAB memory bits	104	
20	Total block memory bits	0 / 7,024,640	0 %
21	Total block memory implementation bits	0 / 7,024,640	0 %
22			
23	Total DSP Blocks	0 / 156	0 %

Conclusiones

- Arquitecturas secuenciales aptas para aplicaciones en los que los requisitos de velocidad no sean muy exigentes
- Arquitecturas semi-paralelas cuando necesitemos aumentar el throughput

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Arquitecturas Secuenciales

Procesado Digital de la Señal en FPGA

Máster Universitario en Ingeniería de Sistemas Electrónicos