Prueba 2: Procesado digital de la señal con FPGAs

Alumno: 10/5/2019

Se desea diseñar un filtro diezmador por 5 para reducir la tasa de muestreo a f_{SL} =50 MHz de una señal capturada a f_{SH} =250 MHz. El filtro diezmador tiene 45 coeficientes, ganancia 4, y su respuesta en frecuencia y respuesta al impulso se muestran en las figuras 1 y 2. Para la implementación del filtro se propone la arquitectura hardware que se muestra en la figura 3.

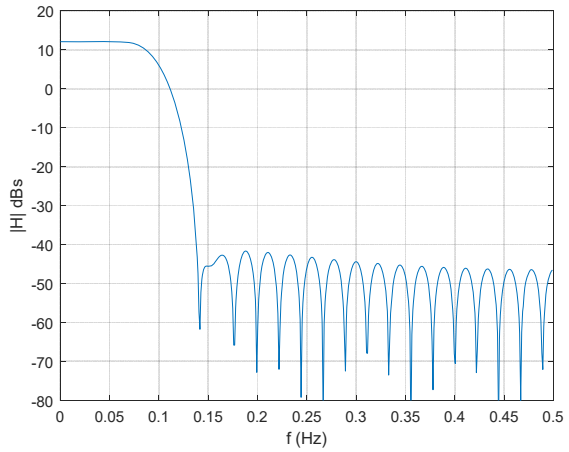

Figura 1. Respuesta en frecuencia del filtro

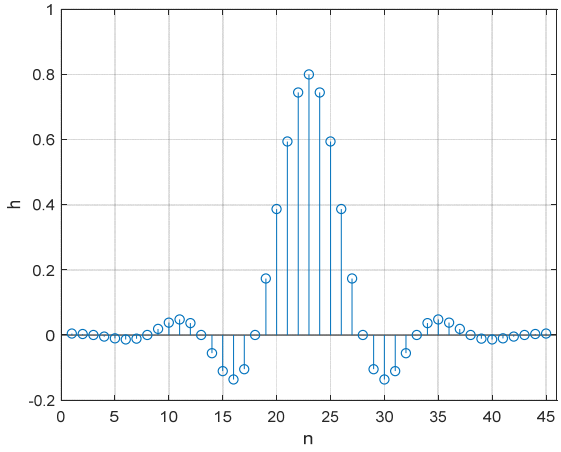

Figura 2. Respuesta al impulso

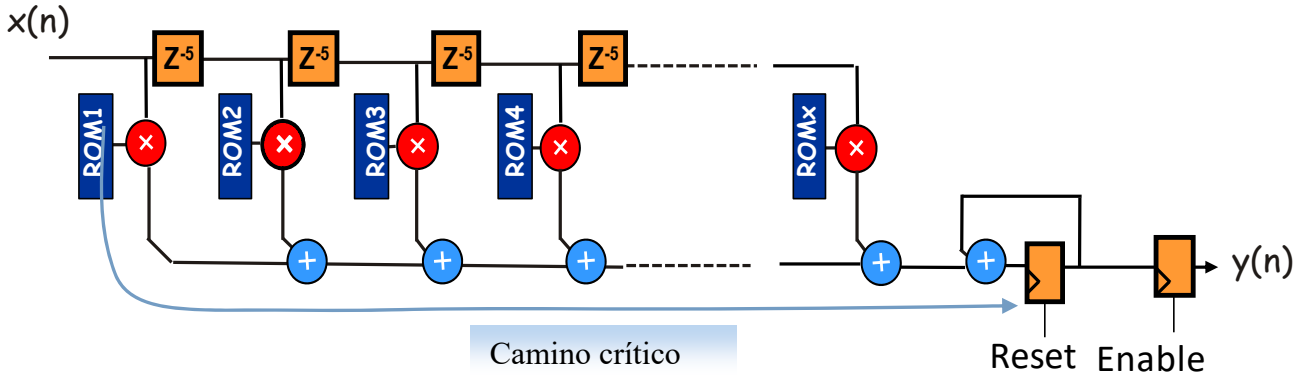

Figura 3. Arquitectura Polifásica del filtro diezmador

1) El filtro está formado por x etapas básicas formadas por un retardo de 5 ciclos, una ROM, un multiplicador y un sumador. ¿Cuántas etapas se requieren para la implementación del filtro? Justifique su respuesta. (1 punto)

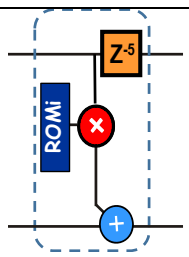

Figura 4. Celda básica

La arquitectura de la figura 3 deriva de una estructura polifásica de un filtro diezmador por 5, que consta de cinco bancos de filtros con un sumador a la salida (ver figura 4.a). Cada subfiltro Hx poseerá 45/5= 9 coeficientes.
Se pueden condensar los 5 bancos en uno sólo, llegando a la estructura de la figura 3. De este modo se necesitarán ***9 etapas*** *como la mostrada en la figura 4 para implementar el filtro completo.*

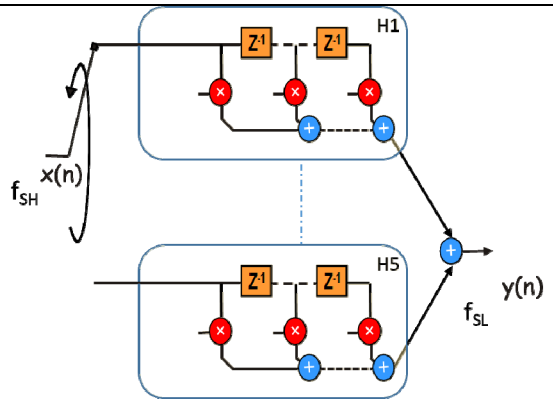

Figura 4.a

2) Indique a qué frecuencia debe funcionar cada uno de los componentes del filtro. Explique para qué sirven las señales de control Reset y Enable. ¿Qué secuencia deberá seguir cada una de ellas? (1.5 puntos)

Cada una de las etapas básicas (figura 4) y el acumulador final funcionarán a la frecuencia f_{SH}=250 MHz. El registro de salida capturará un dato cada f_{SL} = 250MHz/5 = 50MHz. De este modo el circuito está haciendo el filtrado y el diezmado.

La señal Reset sirve para limpiar el acumulador cuando termina de computar un resultado. Su secuencia deberá ser: [0 0 0 0 1]

La señal de Enable sirve para capturar el dato de salida. Se activará cuando el dato de salida se haya computado. [0 0 0 0 1]

3) Indique qué coeficientes se almacenan en cada memoria del filtro. (1punto)

Harán falta nueve memorias ROM1…ROM9
Cada memoria almacenará cinco coeficientes.
ROM1: h0 h1 h2 h3 h4
ROM2: h5 … h9
ROM3: h10 …h14
ROM4: h15…h19
ROM5: h20…h24
ROM6: h25…h29
ROM7: h30…h34
ROM8: h35…h39
ROM9: h40…h44

4) Dibuje (sobre el esquema de la figura 3) el camino crítico del circuito y halle la máxima frecuencia de funcionamiento sabiendo que t_{mult}=3.5ns y t_{add}=1.5ns (NOTA: considere nulos los tiempos de set-up y propagación de los registros para resolver este ejercicio). (1 punto)

El camino crítico recorre 1 multiplicador y 9 sumadores.
$t_{crit} = 9*t_{add} + t_{mult} = 9*1.5 + 3.5 = 17$ ns
La frecuencia máxima se calculará teniendo en cuenta el camino crítico
→ $f_{max} = 1/t_{crit} = 57.8$ MHz << 250 MHz

Dado que la frecuencia máxima es menor de 250MHz, este filtro no cumple con los requisitos.

5) Proponga la segmentación adecuada para implementar el filtro diezmador en un dispositivo FPGA Virtex 7 y que pueda funcionar a la frecuencia de reloj de 250 MHz. Tenga en cuenta que el dispositivo Virtex 7 dispone de bloques DSP48, cuyo esquema se muestra a continuación. Indique cual es la latencia adicional del filtro una vez segmentado. (1.5 puntos)

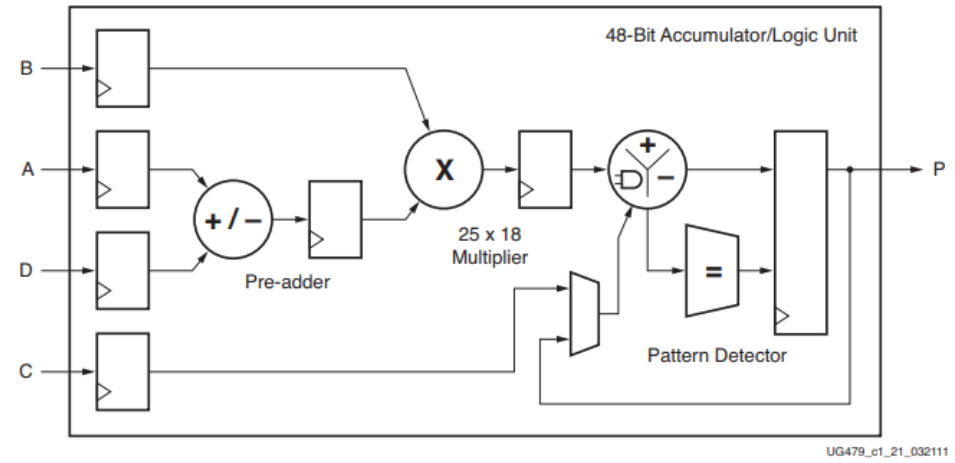

Figura 5. Estructura del DSP48E1 (Virtex-7)

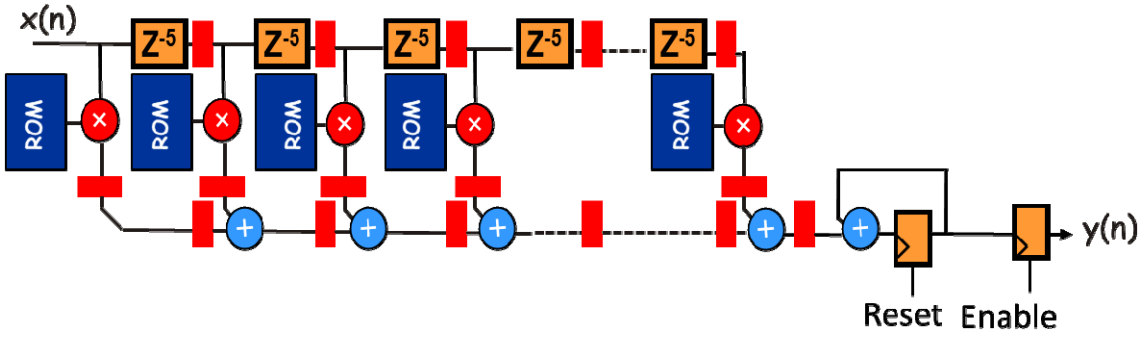

Figura 6. Estructura del filtro diezmador

Se puede segmentar:

- *Horizontalmente: añadiendo registros a la salida de los multiplicadores*
- *Verticalmente: añadiendo registros en la línea de datos y en la cadena de sumadores*

Tras segmentar la frecuencia máxima será fmax=1/3.5 = 285.7MHz > 250MHz. De este modo sí que cumple con los requisitos.

Para calcular la latencia tenemos que tener en cuenta el número de ciclos de retardo entre la entrada y la salida:

- *1 ciclo debido al registro de detrás de los multiplicadores + 9 ciclos debidos a los registros de la cadena de sumadores + 5 ciclos del acumulador = 15 ciclos de latencia*
- *Nótese que la primera etapa se ha segmentado (verticalmente) por mantener la regularidad del circuito, pero no es necesario segmentarla*

*En unidades de tiempo la latencia será t_{lat} = 15*1/250MHz = 60ns*

6) Se desea cuantificar la entrada de datos y los coeficientes en complemento a dos con el formato [10,9] con signo. Se debe cuantificar la salida de forma que se evite el desbordamiento y se mantengan 9 bits fraccionales de precisión a la salida, mientras que internamente el filtro deberá operar con precisión completa. Cuantifique todos los operadores internos del filtro. Si en alguna conexión entre operadores se requiere cambiar el formato indíquelo con un bloque Q y escriba el nuevo formato a su salida. (1.5 puntos)

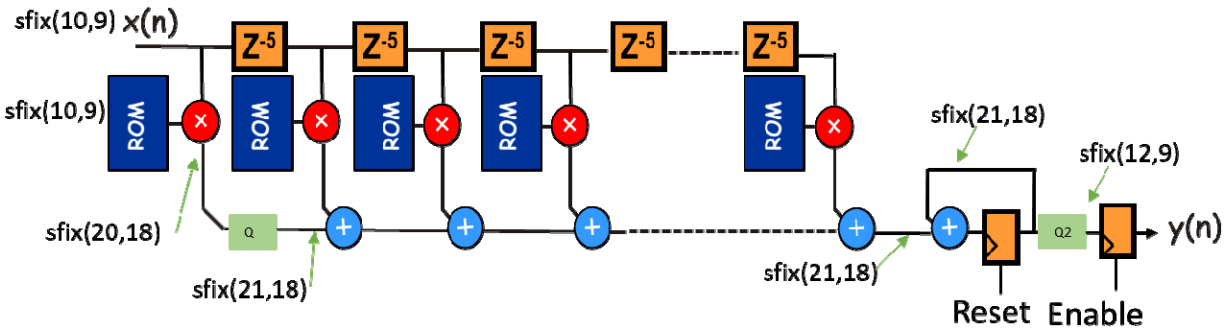

Figura 5. Estructura del filtro

La salida de los multiplicadores tendrá formato sfix(20,18).
Dado que la ganancia del circuito es 4 se necesitarán 2 bits de guarda, por lo que la cadena de sumadores la podemos cuantificar con 21 bits y formato sfix(21,18).
Ajustamos la cuantificación a la salida a la precisión pedida (9 bits), con tres bits de parte entera porque la ganancia es 4. Por lo tanto el formato de salida será sfix(12,9).

El cuantificador Q1 deberá realizar la extensión del bit de signo para pasar de 20 a 21 bits, mientras que el cuantificador Q2 cogerá los bits [20…9].

7) Suponiendo que tenemos codificado en Verilog el módulo mostrado en la figura 8, ¿cómo realizaría la instanciación de M módulos de forma eficiente? Complete el código verilog proporcionado para instanciar los M módulos. (1.5 puntos)

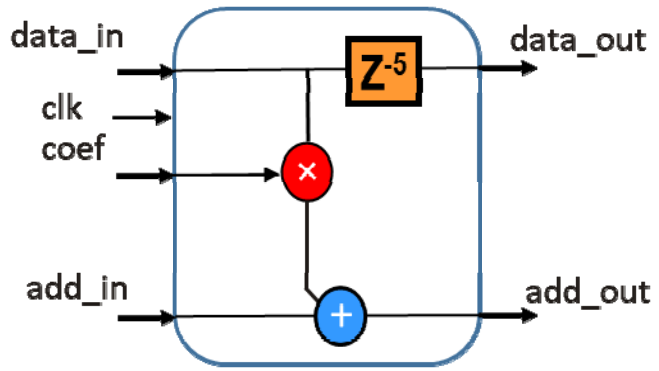

Figura 8. Módulo de la celda básica Mult_add

```
module Mult_add
#(   parameter Win,
     parameter Wout,
     parameter Wc)
 ( input signed [Win-1:0] data_in,
   input signed [Wout-1:0] add_in,
   input signed [Wc-1:0] coef,
   input clk,
   output signed [Win-1:0] data_out,
   output signed [Wout-1:0] add_out );
```

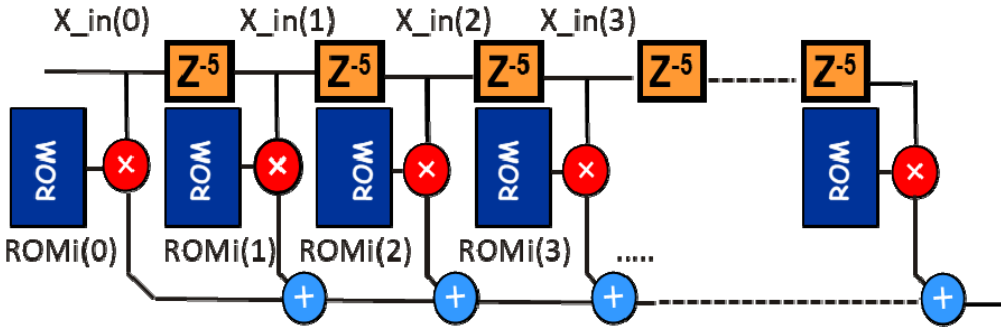

Figura 9. Conexionado de las celdas básicas

```
wire signed [Win-1:0] X_in[M-1  :0];
wire signed [Wc-1:0] ROMi[M-1:0];
wire signed [Wout-1:0] addi[M:0];


assign data[0] = X_in[0];
assign addi[0] = {Win-1{1'b0}};

genvar i;

generate
        for (i=0;i<M;i=i+1)
                begin: Madds
                        Mult_add tap_Madd(.data_in(data[i]), .add_in(addi[i]), .coef(ROMi[i]), .clk(clk), .data_out(data[i+1]), .add_out(addi[i+1]);
        end
endgenerate
```

La forma más eficiente de instanciar los M módulos será utilizando el comando generate. Para ello deberemos realizar un bucle "for" instanciando los M módulos y conectando las señales de forma que la salida de un módulo se conecte a la entrada del siguiente.

8) Codificar, utilizando el lenguaje de descripción hardware Verilog, la celda básica de la figura 10. (1.5 puntos)

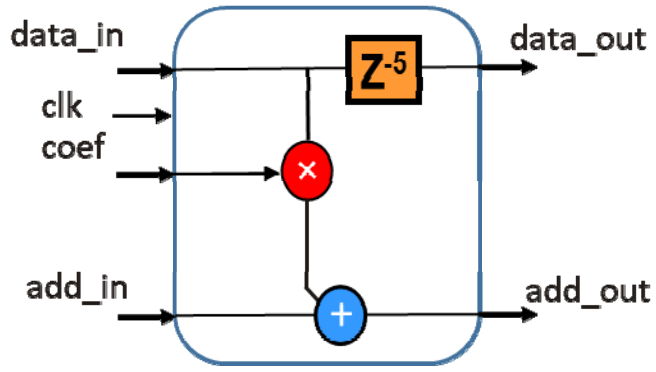

Figura 10. Módulo de la celda básica Mult_add

```
module Mult_add
#(   parameter Win,
     parameter Wout,
    parameter Wc)
 ( input signed [Win-1:0] data_in,
   input signed [Wout-1:0] add_in,
   input signed [Wc-1:0] coef,
   input clk,
   output signed [Win-1:0] data_out,
   output signed [Wout-1:0] add_out );
```

```
module Mult_add
#(   parameter Win=10,
     parameter Wout=22,
    parameter Wc=10)
 ( input signed [Win-1:0] data_in,
   input signed [Wout-1:0] add_in,
   input signed [Wc-1:0] coef,
   input clk,
   output signed [Win-1:0] data_out,
   output signed [Wout-1:0] add_out );

  reg signed [Win-1:0] data_reg[5:0];
  wire signed [Win+Wc-1:0] mult;

  integer i;

   always@(posedge clk)
       begin
            data_reg[0] = data_in;
           for(i = 1; i <= 5; i=i+1) begin
                    data_reg[i] <= data_reg[i-1];
           end
        end

     assign mult = coef * data_in;
     assign add_out = add_in + mult;
    assign data_out = data_reg[5];

endmodule
```