Filtros FIR

May 16, 2024

1 Notación

Asumiremos que un vector binario x tiene la siguiente forma:

x_{B-1}	x_{B-2}	$\cdots$	x_3	x_2	x_1	x_0

Sea x una señal de tiempo discreto que es codificada con B bits. Por lo tanto, cada muestra de dicha señal se notará $x[n]$ y cada bit particular de la muestra se notará $x_i[n]$, $i = 0, 1, \ldots, B-1$, $n \in \mathbb{Z}$.

2 Introducción

Un filtro FIR es un sistema LTI (linear - time invariance) el cual relaciona su entrada x con su salida y de la siguiente forma:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] \tag{1}$$

donde $h[k]$ son constantes tales que $h[0] \neq 0$, $h[L-1] \neq 0$.

Equivalentemente, existen distintas notaciones para la ec. (1):

$$\begin{aligned} y[n] &= \textstyle\sum_{k=0}^{L-1} h[k]x[n-k] \\ y[n] &= h[n] * x[n] \\ y[n] &= \mathbf{h^T x_n} \end{aligned} \tag{2}$$

donde

$$\mathbf{h} = \begin{pmatrix} h[0] \\ h[1] \\ \vdots \\ h[L-1] \end{pmatrix} \tag{3}$$

$$\mathbf{x_n} = \begin{pmatrix} x[n] \\ x[n-1] \\ \vdots \\ x[n-L+1] \end{pmatrix} \tag{4}$$

3 Propiedades básicas

3.1 Respuesta impulsiva

Sea la entrada al sistema:

$$x[n] = \delta[n] = \begin{cases} 1 & \text{Si } n = 0 \\ 0 & \text{Si } n \neq 0 \end{cases} \tag{5}$$

Entonces $y[n] = h[n]$ es la respuesta impulsiva del filtro.

3.2 Máximo valor a la salida

Si $|x[n]| \leq x_{max}$, para todo n, entonces

$$|y[n]| \leq x_{max} \sum_{k=0}^{L-1} |h[k]| \tag{6}$$

3.3 Respuesta en frecuencia

Aplicando Transformada Z a la ec. (1), se obtiene

$$Y(z) = Z\{h[n] * x[n]\} = H(z)X(z) \tag{7}$$

donde

$$H(z) = \sum_{k=0}^{L-1} h[k]z^{-k} = h[0] + h[1]z^{-1} + \ldots + h[L-1]z^{-L+1} \tag{8}$$

- Observar que $H(z)$ no tiene polos, y sólo tiene ceros en las raízes del polinomio.
- Observar que el filtro siempre es estable, es decir, no existe valor de z para el cual $H(z)$ no sea acotado.

Sean:

$$H(\Omega) = H(z)\lfloor_{z=e^{j\Omega}} = \sum_{k=0}^{L-1} h[k]e^{j\Omega} \tag{9}$$

$$\phi(\Omega) = \arctan\left(\frac{\text{Imag}[H(\Omega)]}{\text{Real}[H(\Omega)]}\right) \tag{10}$$

Llamamos:

- $|H(\Omega)|$ respuesta de la magnitud en frecuencia.
- $\phi(\Omega)$ respuesta de la phase en frecuencia.

3.4 Simetría, antisimetría y fase lineal

Un filtro FIR es **simétrico** si cumple la condición:

$$\begin{array}{rcl} h[0] & = & h[L-1] \\ h[1] & = & h[L-2] \\ & \vdots & \\ h[i] & = & h[L-1-i] \end{array} \tag{11}$$

Un filtro FIR es **antisimétrico** si cumple la condición:

$$\begin{array}{rcl} h[0] & = & -h[L-1] \\ h[1] & = & -h[L-2] \\ & \vdots & \\ h[i] & = & -h[L-1-i] \end{array} \tag{12}$$

Hay cuatro casos:

1. L es par, filtro simétrico:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] = \sum_{k=0}^{L/2-1} h[k](x[n-k] + x[n-L+k+1]) \tag{13}$$

2. L es par, filtro antisimétrico:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] = \sum_{k=0}^{L/2-1} h[k](x[n-k] - x[n-L+k+1]) \tag{14}$$

3. L es impar, filtro simétrico:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] = h[(L-1)/2]x[n-(L-1)/2] + \sum_{k=0}^{(L-1)/2-1} h[k](x[n-k] + x[n-L+k+1]) \tag{15}$$

4. L es impar, filtro antisimétrico:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] = h[(L-1)/2]x[n-(L-1)/2] + \sum_{k=0}^{(L-1)/2-1} h[k](x[n-k] - x[n-L+k+1]) \tag{16}$$

La condición de fase lineal implica que el retardo de grupo D es constante, es decir:

$$D = \frac{d\phi(\Omega)}{d\Omega} = \text{ constante.} \tag{17}$$

Supongamos L par, filtro simétrico. Entonces

$$\begin{aligned}
H(z) &= \textstyle\sum_{k=0}^{L-1} h[k]z^{-k} \\
&= h[0] + h[1]z^{-1} + \ldots + h[L-1]z^{-L+1} \\
&= h[0] + h[1]z^{-1} + \ldots + h[1]z^{-L+2} + h[0]z^{-L+1} \\
&= h[0](1 + z^{-L+1}) + h[1](z^{-1} + z^{-L+2}) + h[2](z^{-2} + z^{-L+3}) + \ldots + h[L/2-1](z^{-L/2+1} + z^{-L/2}) \\
&= \textstyle\sum_{k=0}^{L/2-1} h[k](z^{-k} + z^{-L+k+1}) \\
&= \textstyle\sum_{k=0}^{L/2-1} h[k]z^{(L-1)/2}z^{-(L-1)/2}(z^{-k} + z^{-L+k+1}) \\
&= \textstyle\sum_{k=0}^{L/2-1} h[k]z^{-(L-1)/2}(z^{(L-1)/2-k} + z^{-(L-1)/2+k})
\end{aligned} \tag{18}$$

Entonces,

$$\begin{aligned}
H(\Omega) = H(z)\big|_{z=e^{j\Omega}} &= \textstyle\sum_{k=0}^{L/2-1} h[k]e^{-j\Omega(L-1)/2}\left(e^{j\Omega((L-1)/2-k)} + e^{-j\Omega((L-1)/2-k)}\right) \\
&= \textstyle\sum_{k=0}^{L/2-1} 2h[k]\cos(\Omega\frac{L-1}{2} - k)e^{-j\Omega(L-1)/2} \\
&= 2e^{-j\Omega(L-1)/2}\textstyle\sum_{k=0}^{L/2-1} h[k]\cos(\Omega\frac{L-1}{2} - k) \\
&= H_R(\Omega)e^{-j\Omega(L-1)/2}
\end{aligned} \tag{19}$$

donde

$$H_R(\Omega) = 2\sum_{k=0}^{L/2-1} h[k]\cos(\Omega\frac{L-1}{2} - k) \tag{20}$$

Observar que $H_R(\Omega)$ es una función real para todo $0 \leq \Omega < 2\pi$ y por lo tanto,

$$\phi(\Omega) = -\Omega\frac{L-1}{2} + \pi\frac{1 - \text{signo}(H_r(\Omega))}{2} \tag{21}$$

Observar que el término $\pi\frac{1-\text{signo}(H_r(\Omega))}{2}$ vale 0 si $H_R(\Omega) \geq 0$, y vale π si $H_R(\Omega) < 0$. Es decir que se agrega π (180°) en el último caso.

Finalmente,

$$D = \frac{d\phi(\Omega)}{d\Omega} = \frac{1-L}{2}, \quad \text{constante.}$$

Demostrar que los otros tres casos (L impar - simétrico, L par - antisimétrico, L impar antisimétrico) son también de fase lineal queda para el lector.

4 Método de diseño de filtros FIR

Sólo mencionamos nombres de los métodos como referencia:

- Ventaneo de la respuesta impulsiva.
- Equiripple o conocido también como minimax.
- Cuadrados mínimos.

5 Implementacióm Mulitiply-and-accumulate (MAC)

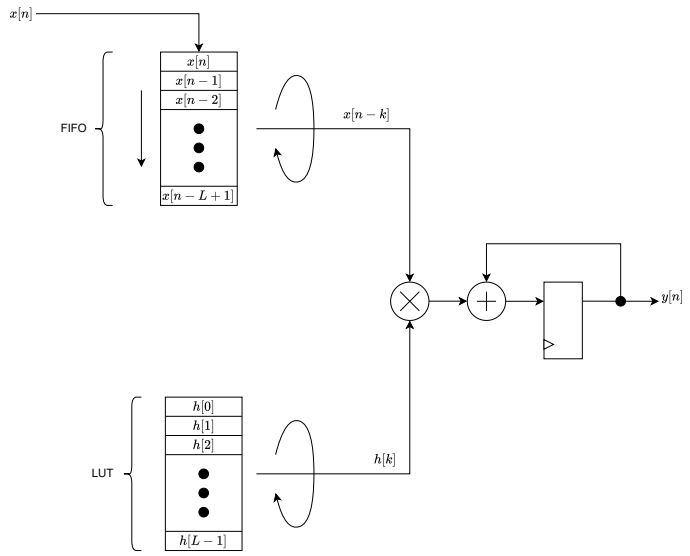

Figure 1: MAC FIR.

Detalles de implementación:

- La FIFO que almacena x en general se implementa de forma circular para evitar el movimiento de todos los datos almacenados en la FIFO a una tasa f_s de forma tal de salvar potencia.
- En general, la entrada $x[n]$ es representada en 2C con B_x bits lo cual implica que $x_{min} = -2^{B_x-1}$, $x_{max} = 2^{B_x-1} - 1$
- La cantidad de bits B_h utilizados para representar $h[k]$ viene dada por:
  - Si todos los $h[k]$ son positivos:

$$h_{max} = \max\{h[h]\}_{k=0,1,\ldots,L-1} \tag{22}$$

$$B_h = \lceil \log_2(h_{max}) \rceil \tag{23}$$

  - Si todos los $h[k]$ son negativos: no tiene sentido implementar todos los $h[k]$ negativos, en lugar de esto, se implementan todos positivos ya que la única diferencia es la inversión de signo en la salida.
  - Si $h[k]$ tiene coeficientes positios y negativos, entonces

$$h_{max} = \max\{|h[h]|\}_{k=0,1,\ldots,L-1} \tag{24}$$

$$B_h = \lceil \log_2(h_{max}) \rceil + 1 \tag{25}$$

- El rango de valores de la salida $y_{min} \leq y[n] \leq y_{max}$ está dado por:

$$\begin{aligned} y_{max} &= x_{max} \textstyle\sum_{k=0}^{L-1} |h[k]| \leq x_{max} h_{max} L = +(2^{B_x-1} - 1)(2^{B_h} - 1)L \\ y_{min} &= x_{min} \textstyle\sum_{k=0}^{L-1} |h[k]| \geq x_{min} h_{max} L = -2^{B_x-1}(2^{B_h} - 1)L \end{aligned} \tag{26}$$

Por lo tanto la cantidad de bits que se necesitan a la salida está dada por

$$B_y = B_x + B_h + \lceil \log_2 L \rceil \tag{27}$$

- Se requieren $B_x + B_h$ bits para el resultado del producto $x[n-k] \times h[k]$ (donde B_x es la cantidad de bits $x[n]$ y B_h es la cantidad de bits de $h[k]$). Comúnmente, se descartan K LSBs para reducir el tamaño, lo que es equivalente a multiplicar por 2^{-K}, es decir que el producto se realiza con $B_p = B_x + B_h - K$ bits y la salida $y[n]$ queda escalada por 2^{-K}.
- Tasa de muestreo:

$$f_s = \frac{f_{clk}}{L} \tag{28}$$

Si además se utiliza un multiplicador sequencial el cual toma B_h ciclos de reloj en obtener una multiplicación, la tasa de muestreo viene dada por:

$$f_s = \frac{f_{clk}}{B_h.L} \tag{29}$$

- Con el objetivo de reducir la cantidad de operaciones, los filtros FIR se implementan la mayoría de las veces con L par y simétricos o antisimétricos. Es más, mayormente, L es potencia de 2.

Ejercicio
Implementar con un multiplicador sequencial de Booth, el filtro FIR que posee los siguientes coeficientes. El filtro es simétrico, $L = 16$. Los bits de entrada y salida son $B_x = B_y = 16$ bits.

k	$h[k]$
0 / 15	10
1 / 14	21
2 / 13	32
3 / 12	43
4 / 11	54
5 / 10	65
6 / 9	76
7 / 8	87

6 Implementacióm por medio de distributed arithmetic (DA)

Se representa $x[n]$ en 2C:

$$x[n] = -x_{B_x-1}[n]2^{B_x-1} + \sum_{i=0}^{B_x-2} x_i[n]2^i \tag{30}$$

donde $x_i[n]$ es el i-ésimo bit de $X[n]$.

Entonces

$$\begin{aligned} y[n] &= \textstyle\sum_{k=0}^{L-1} h[k]x[n-k] \\ &= \textstyle\sum_{k=0}^{L-1} h[k]\left\{-x_{B_x-1}[n-k]2^{B_x-1} + \sum_{i=0}^{B_x-2} x_i[n-k]2^i\right\} \\ &= -2^{B_x-1}\left\{\textstyle\sum_{k=0}^{L-1} h[k]x_{B_x-1}[n-k]\right\} + \sum_{i=0}^{B_x-2} 2^i\left\{\sum_{k=0}^{L-1} h[k]x_i[n-k]\right\} \end{aligned} \tag{31}$$

Observar que $x_i[n-k] \in \{0,1\}$, entonces $\sum_{k=0}^{L-1} h[k]x_i[n-k]$ puede adoptar 2^L valores diferentes. Entonces como los $h[k]$ son constantes, se puede precomputar y almacenar en una LUT. No se necesita de un multiplicador.

Ejemplo

k	$h[k]$
0	0011 ($+3_{10}$)
1	1110 (-2_{10})
2	0111 ($+7_{10}$)
3	1011 (-5_{10})

Entonces $x_i[n-k]$ puede valor 0 ó 1 para $i = 0, 1, \ldots, B_x - 1$, por lo cual el contenido de la LUT será el siguiente:

$x_i[n-3]$	$x_i[n-2]$	$x_i[n-1]$	$x_i[n]$	$\sum_{k=0}^{L-1} h[k]x_i[n-k]$	Valor	LUT
0	0	0	0	0	0_{10}	00000_{2C}
0	0	0	1	$h[0]$	$+3_{10}$	00011_{2C}
0	0	1	0	$h[1]$	-2_{10}	11110_{2C}
0	0	1	1	$h[1] + h[0]$	$+1_{10}$	00001_{2C}
0	1	0	0	$h[2]$	$+7_{10}$	00111_{2C}
0	1	0	1	$h[2] + h[0]$	$+10_{10}$	01010_{2C}
0	1	1	0	$h[2] + h[1]$	$+4_{10}$	00100_{2C}
0	1	1	1	$h[2] + h[1] + h[0]$	$+8_{10}$	01000_{2C}
1	0	0	0	$h[3]$	-5_{10}	11011_{2C}
1	0	0	1	$h[3] + h[0]$	-2_{10}	11100_{2C}
1	0	1	0	$h[3] + h[1]$	-7_{10}	11001_{2C}
1	0	1	1	$h[3] + h[1] + h[0]$	-4_{10}	11100_{2C}
1	1	0	0	$h[3] + h[2]$	$+2_{10}$	00010_{2C}
1	1	0	1	$h[3] + h[2] + h[0]$	$+5_{10}$	00101_{2C}
1	1	1	0	$h[3] + h[2] + h[1]$	0_{10}	00000_{2C}
1	1	1	1	$h[3] + h[2] + h[1] + h[0]$	$+3_{10}$	00011_{2C}

En la Fig. 2 se observa la implementación del filtro con DA.

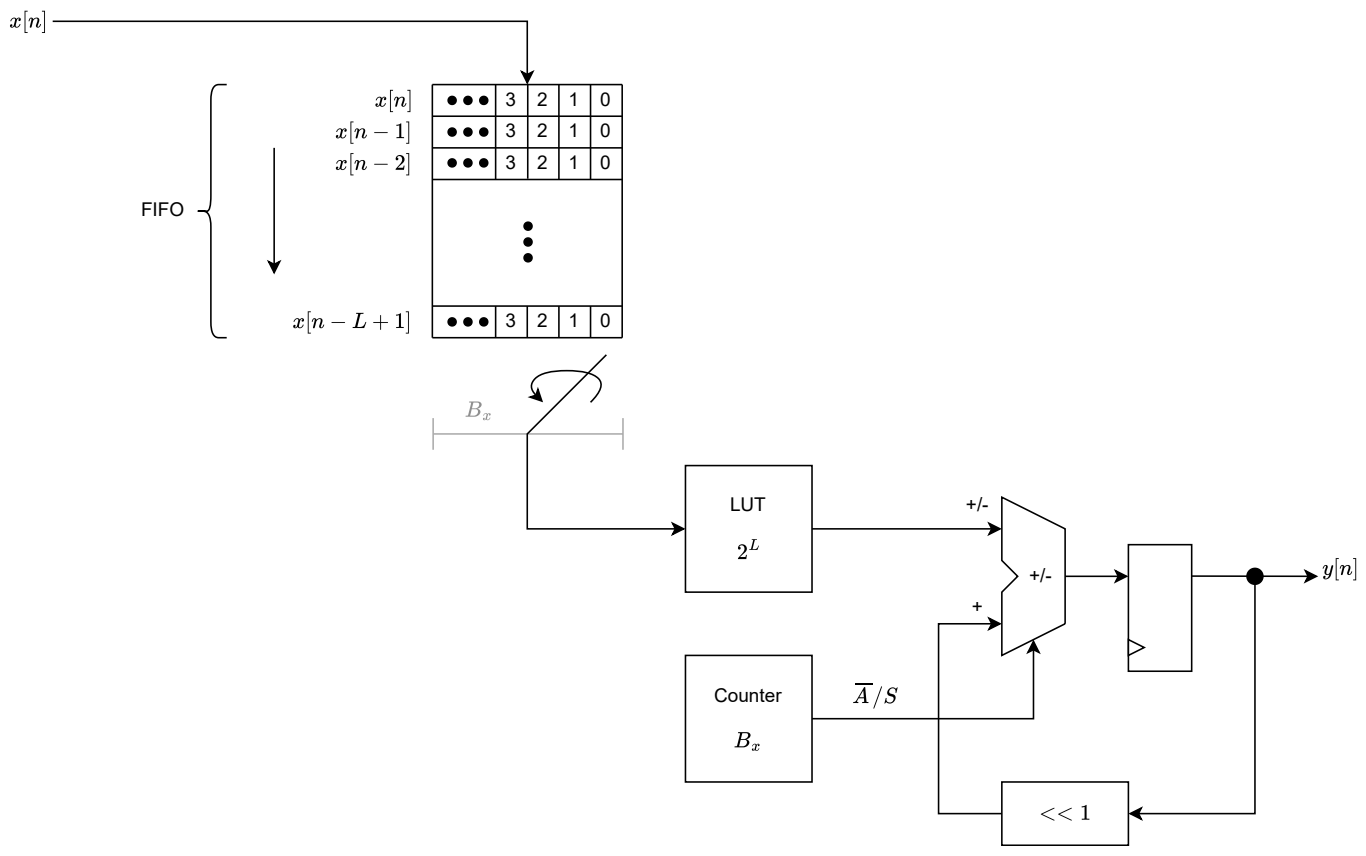

Figure 2: DA FIR.

Se puede realizar una optimización adicional para reducir la LUT a la mitad de palabras. Para tal fin se expresa cada muestra de $x[n]$ de la siguiente forma:

$$x[n-k] = \frac{x[n-k] + x[n-k]}{2} = \frac{x[n-k] - (-x[n-k])}{2} \tag{32}$$

Redordando la expresión 2C:

$$x[n-k] = -2^{Bx-1} x_{B_x-1}[n-k] + \sum_{i=0}^{B_x-2} x_i[n-k]2^i \tag{33}$$

Entonces, en 2C:

$$-x[n-k] = \overline{x}[n-k] + 1 = -2^{Bx-1}\overline{x_{B_x-1}}[n-k] + \sum_{i=0}^{B_x-2} \overline{x_i}[n-k]2^i + 1 \tag{34}$$

Por lo tanto, la ec. (32) se puede escribir como:

$$\begin{aligned} x[n-k] &= \tfrac{x[n-k]-(-x[n-k])}{2} \\ &= \tfrac{\left(-2^{Bx-1}x_{B_x-1}[n-k]+\sum_{i=0}^{B_x-2} x_i[n-k]2^i\right)-\left(-2^{Bx-1}\overline{x_{B_x-1}}[n-k]+\sum_{i=0}^{B_x-2}\overline{x_i}[n-k]2^i+1\right)}{2} \\ &= \tfrac{-2^{Bx-1}(x_{B_x-1}[n-k]-\overline{x_{B_x-1}}[n-k])+\sum_{i=0}^{B_x-2}(x_i[n-k]-\overline{x_i}[n-k])2^i-1}{2} \\ &= \tfrac{1}{2}\left[-2^{Bx-1}(x_{B_x-1}[n-k]-\overline{x_{B_x-1}}[n-k])+\sum_{i=0}^{B_x-2}(x_i[n-k]-\overline{x_i}[n-k])2^i-1\right] \end{aligned} \tag{35}$$

Para simplificar la notación, se escribe:

$$x_i[n-k] - \overline{x_i}[n-k] = x_i^*[n-k] = \begin{cases} 1 & \text{si } x_i[n-k] = 1 \\ \overline{1} & \text{si } x_i[n-k] = 0 \end{cases} \tag{36}$$

Observar que $x^*[n-k]$ queda en codificación SD. Entonces,

$$x[n-k] = \frac{1}{2}\left[-2^{Bx-1}x^*_{B_x-1}[n-k] + \sum_{i=0}^{B_x-2} x_i^*[n-k]2^i - 1\right] \tag{37}$$

Recordando la salida del filtro, entonces:

$$\begin{aligned} y[n] &= \textstyle\sum_{k=0}^{L-1} h[k]x[n-k] \\ &= \tfrac{1}{2}\textstyle\sum_{k=0}^{L-1} h[k]\left[-2^{Bx-1}x^*_{B_x-1}[n-k] + \sum_{i=0}^{B_x-2} x_i^*[n-k]2^i - 1\right] \\ &= \tfrac{1}{2}\left[-2^{B_x-1}\textstyle\sum_{k=0}^{L-1} h[k]x^*_{B_x-1}[n-k] + \sum_{i=0}^{B_x-2} 2^i\left(\sum_{k=0}^{L-1} h[k]x_i^*[n-k]\right) - \sum_{k=0}^{L-1} h[k]\right] \end{aligned} \tag{38}$$

Observar que $x_i^*[n-k] \in \{\overline{1}, 1\}$, entonces $\sum_{k=0}^{L-1} h[k]x_i^*[n-k]$ puede adoptar nuevamente 2^L valores diferentes. Sin embargo, ahora exite una ventaja adicional: la segunda mitad de la tabla es equivalente a la primera mitad pero reflejada ycon el signo invertido, con lo cual puede simplificarse la LUT a mitad del tamaño. Siguiendo el ejemplo:

$x_i[n-3]$	$x_i[n-2]$	$x_i[n-1]$	$x_i[n]$	$\sum_{k=0}^{L-1} h[k]x_i^*[n-k]$	Valor	LUT
0	0	0	0	$-(h[3]+h[2]+h[1]+h[0])$	-3_{10}	111101_{2C}
0	0	0	1	$-(h[3]+h[2]+h[1]-h[0])$	$+3_{10}$	000011_{2C}
0	0	1	0	$-(h[3]+h[2]-h[1]+h[0])$	-7_{10}	111001_{2C}
0	0	1	1	$-(h[3]+h[2]-h[1]-h[0])$	-1_{10}	111111_{2C}
0	1	0	0	$-(h[3]-h[2]+h[1]+h[0])$	$+11_{10}$	001011_{2C}
0	1	0	1	$-(h[3]-h[2]+h[1]-h[0])$	$+17_{10}$	010001_{2C}
0	1	1	0	$-(h[3]-h[2]-h[1]+h[0])$	$+7_{10}$	000111_{2C}
0	1	1	1	$-(h[3]-h[2]-h[1]-h[0])$	$+13_{10}$	001101_{2C}
1	0	0	0	$+(h[3]-h[2]-h[1]-h[0])$	-13_{10}	000011_{2C}
1	0	0	1	$+(h[3]-h[2]-h[1]+h[0])$	-7_{10}	111101_{2C}
1	0	1	0	$+(h[3]-h[2]+h[1]-h[0])$	-17_{10}	111001_{2C}
1	0	1	1	$+(h[3]-h[2]+h[1]+h[0])$	-11_{10}	000001_{2C}
1	1	0	0	$+(h[3]+h[2]-h[1]-h[0])$	$+1_{10}$	110101_{2C}
1	1	0	1	$+(h[3]+h[2]-h[1]+h[0])$	$+7_{10}$	101111_{2C}
1	1	1	0	$+(h[3]+h[2]+h[1]-h[0])$	-3_{10}	111001_{2C}
1	1	1	1	$+(h[3]+h[2]+h[1]+h[0])$	$+3_{10}$	110011_{2C}

En la Fig. 3 se observa la implementación. $x_i[n-L+1]$ selecciona la mitad que corresponde de la LUT. En el caso de $x_i[n-L+1]=1$, el resto de los bits $x_i[n-L+2],\ldots,x_i[n]$ se invierten para reflejar la LUT. Asimismo, es necesario iniciar la cuenta con el término $\sum_{k=0}^{L-1} h[k]$.

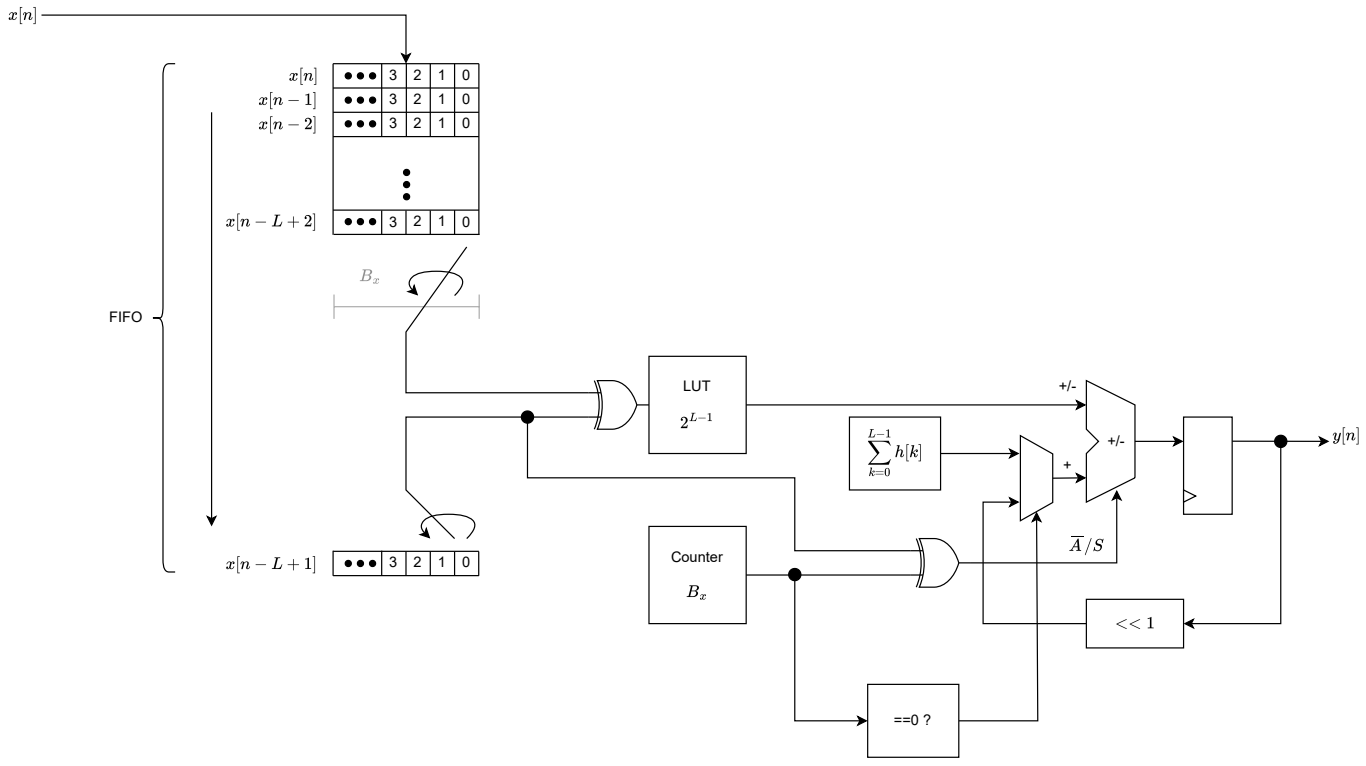

Figure 3: DA FIR con LUT reducida.

Se puede reducir la LUT a la mitad nuevamente tomando no solo el bit $x_i[n-L+1]$ para decidir la operación de suma o resta, sino también el bit $x_i[n-L+2]$. Siguiendo con el mismo ejemplo, se vuelve a partir la LUT de forma simétrica:

$x_i[n-3]$	$x_i[n-2]$	$x_i[n-1]$	$x_i[n]$	$\sum_{k=0}^{L-1} h[k]x_i^*[n-k]$	Valor	LUT
0	0	0	0	$-(h[3]+h[2]+h[1]+h[0])$	-3_{10}	111101_{2C}
0	0	0	1	$-(h[3]+h[2]+h[1]-h[0])$	$+3_{10}$	000011_{2C}
0	0	1	0	$-(h[3]+h[2]-h[1]+h[0])$	-7_{10}	111001_{2C}
0	0	1	1	$-(h[3]+h[2]-h[1]-h[0])$	-1_{10}	111111_{2C}
0	1	0	0	$-(h[3]-h[2]+h[1]+h[0])$	$+11_{10}$	001011_{2C}
0	1	0	1	$-(h[3]-h[2]+h[1]-h[0])$	$+17_{10}$	010001_{2C}
0	1	1	0	$-(h[3]-h[2]-h[1]+h[0])$	$+7_{10}$	000111_{2C}
0	1	1	1	$-(h[3]-h[2]-h[1]-h[0])$	$+13_{10}$	001101_{2C}
1	0	0	0	$+(h[3]-h[2]-h[1]-h[0])$	-13_{10}	000011_{2C}
1	0	0	1	$+(h[3]-h[2]-h[1]+h[0])$	-7_{10}	111101_{2C}
1	0	1	0	$+(h[3]-h[2]+h[1]-h[0])$	-17_{10}	111001_{2C}
1	0	1	1	$+(h[3]-h[2]+h[1]+h[0])$	-11_{10}	000001_{2C}
1	1	0	0	$+(h[3]+h[2]-h[1]-h[0])$	$+1_{10}$	110101_{2C}
1	1	0	1	$+(h[3]+h[2]-h[1]+h[0])$	$+7_{10}$	101111_{2C}
1	1	1	0	$+(h[3]+h[2]+h[1]-h[0])$	-3_{10}	111001_{2C}
1	1	1	1	$+(h[3]+h[2]+h[1]+h[0])$	$+3_{10}$	110011_{2C}

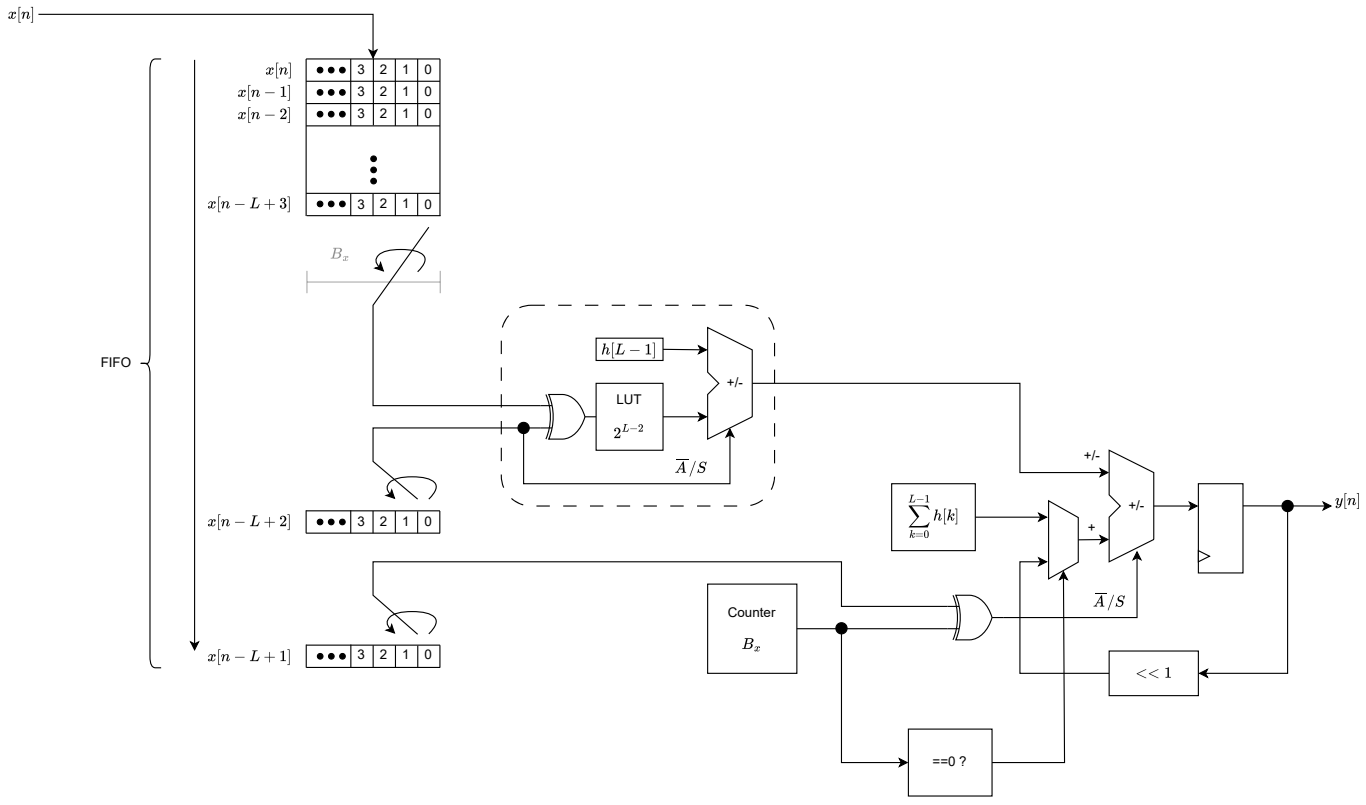

Figure 4: DA FIR con LUT reducida.

Se puede seguir reduciendo la LUT hasta incluso obtener una architectura DA sin LUT (LUT-less DA) a expensas de intercambiar por sumadores. En dicho caso, será necesario un total de L sumadores/restadores para sintetizar el término $\sum_{k=0}^{L-1} h[k]x_i^*[n-k]$.

Queda para el lector continuar con esta idea para deducir el circuito para los casos de simetría/antisimetría y L par/impar.

6.1 Forma Directa (Direct Form - DF)

Las implementaciones MAC y DA intentan reducir la complejidad del hardware a expensas de tardar varios ciclos de reloj en obtener cada muestra $y[n]$.

La forma directa (y en secciones siguientes, la forma transpuesta) intenta tener una velocidad de procesamiento de un sample $y[n]$ por ciclo de reloj.

Recordar la ecuación del filtro FIR:

$$y[n] = \sum_{k=0}^{L-1} h[k]x[n-k] \tag{39}$$

La forma directa corresponde al siguiente diagrama en bloques. El producto por las constante $h[k]$ se puede implementar con CSD para reducir términos.

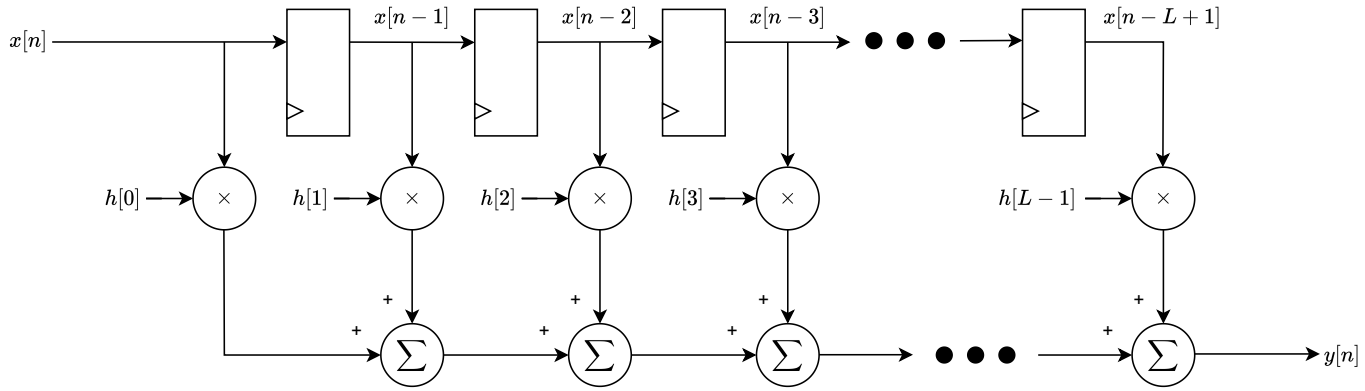

Figure 5: Direct forma FIR.

Comúnmente, la salida $y[n]$ se registra para no tener un camino combinacional entre $x[n]$ e $y[n]$.

Adicionalmente, se puede pensar reducir toda la cadena de sumadores con un árbol de reducción (Wallace/Dadda) y una sola etapa de CPA.

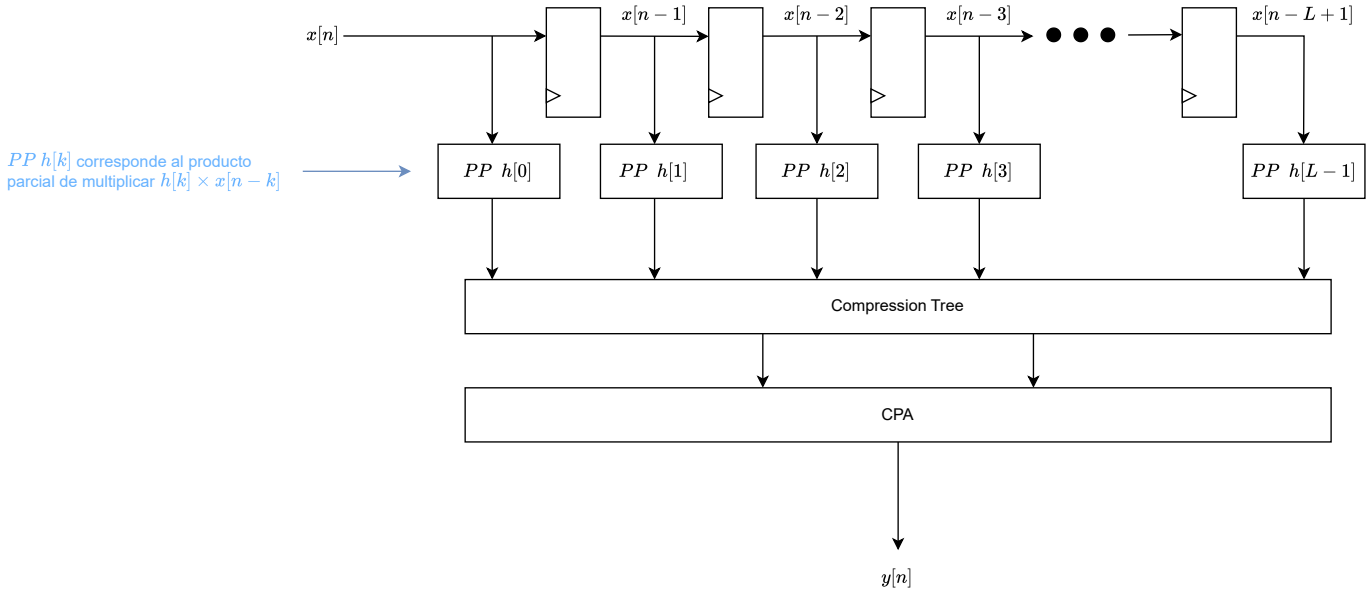

Figure 6: Direct forma FIR con reduction tree.

En la Fig. 7 se observa la implementación de un filtro FIR simétrico de L par.

6.2 Forma transpuesta (Transposed Form - TF)

En la Fig. 8 se observa la forma transpuesta.

La siguientes secciones se trabajará con un filtro no simétrico/antisimétrico. Dichos casos son fáciles de extender y quedará para el lector.

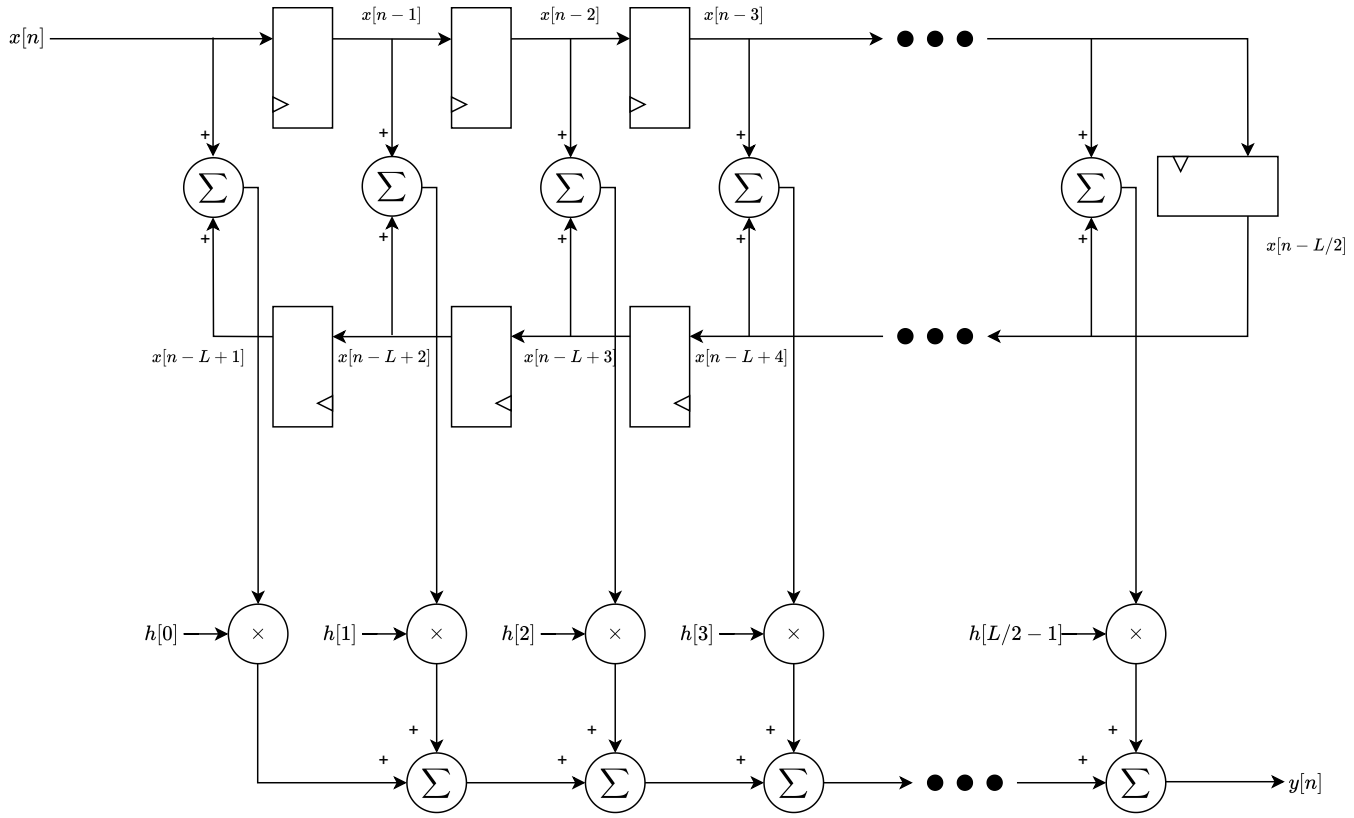

Figure 7: Transposed Form FIR.

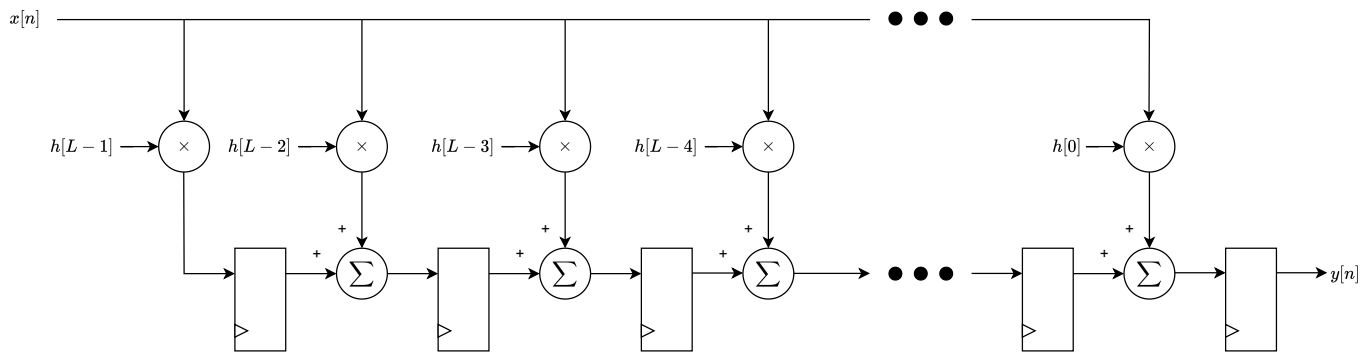

Figure 8: Transposed Form FIR.

En general, un filtro fir en forma transpuesta se divide en dos bloques fundamentales: el multiplier block y el adder block:

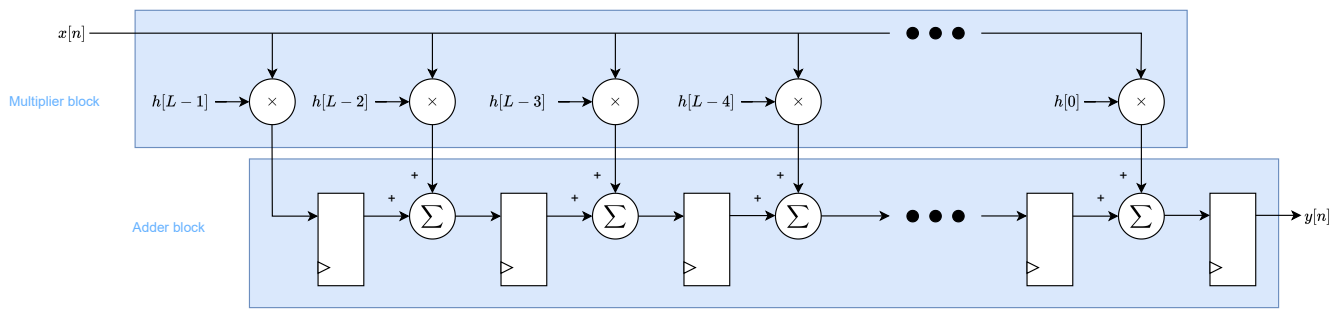

Figure 9: Transposed Form FIR - Adder block, multiplier block.

El adder block puede ser implementado directamente con $L-1$ carry-propagate adders o por medio de compresores y una sola etapa de CPA tal como se observa en la Fig. 10.

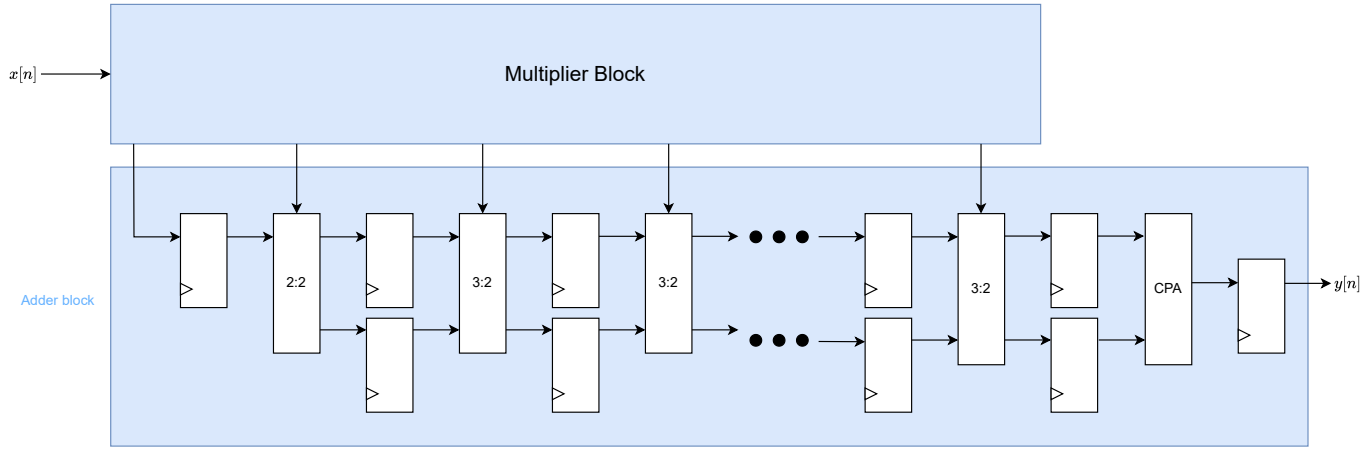

Figure 10: Transposed Form FIR - Adder block implementado con compression.

6.3 Multiple constant multiplication

Observar que en la forma transpuesta, todas las multipliciones por las constantes $h[k]$, $k = 0, 1, \ldots, L-1$, se realiza en el mismo instante de tiempo, es decir se realizan todas sobre la misma muestra $x[n]$. Por ejemplo, sea $L = 2$, $h[0] = 54$, $h[1] = 45$, se puede descomponer cada $h[k]$ en sus factores:

- $h[0] = 54 = 9 \times 6$
- $h[1] = 45 = 9 \times 5$

Se puede descomponer entonces $h[0] = h_c \times 6$, $h[1] = h_c \times 5$, donde $h_c = 9$, lo cual se observa en la Fig. 11.

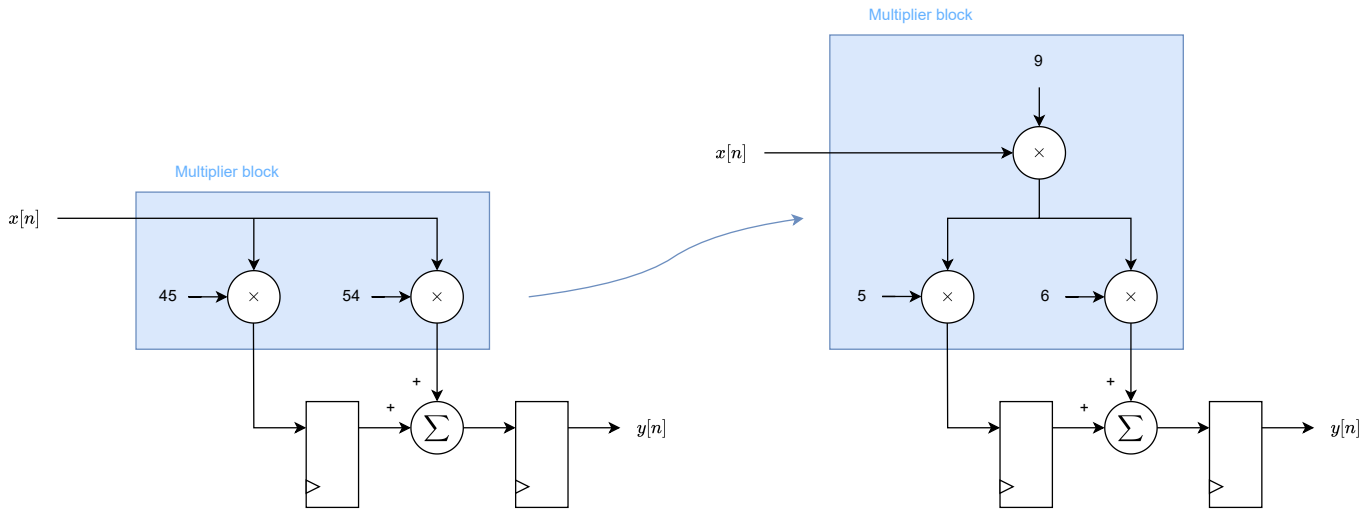

Figure 11: Multiplier block implementado mediante factores comunes.

Esta idea que se puede explotar a nivel de bit es conocida como Common Subexpression Elimination (CSE). El problema es que encontrar la cantidad mínima de términos que componen un bloque multiplicador depende de la codificación empleada (2C, CSD, etc) y resulta en un problema NP-Completo. Por lo cual, distintos métodos heurísticos han sido propuestos. Sólo se presentan aquí algunos y se mencionan ventajas y desventajas. Este enfoque puede ser usado toda vez que una entrada x deba ser multiplicada por un conjunto de coeficientes constantes. Este problema se conoce como Multiple Constant Multiplication y con el objetivo de reducir la cantidad de factores se aplica en la resolución alguno de los métodos de CSE.

6.4 Reduced Adder Graph

Uno de los primeros métodos propuestos en este aspecto [3], [4] fue el de descomponer cada constante en una cantidad mínima de operaciones. Por ejemplo, supongamos la constante $93_{10} = 1011101_2 = 10\bar{1}00\bar{1}01_{CSD}$. Se observa que tiene menos términos la descomposición $x \times 93_{10} = x(1+2)(32-1)$ (dos sumas ya eue la descomposición CSD $93_{10} \times x = 93_{10} \times 1100\bar{1}01_{CSD} = x(64+32-4+1)$ (cuatro sumas).

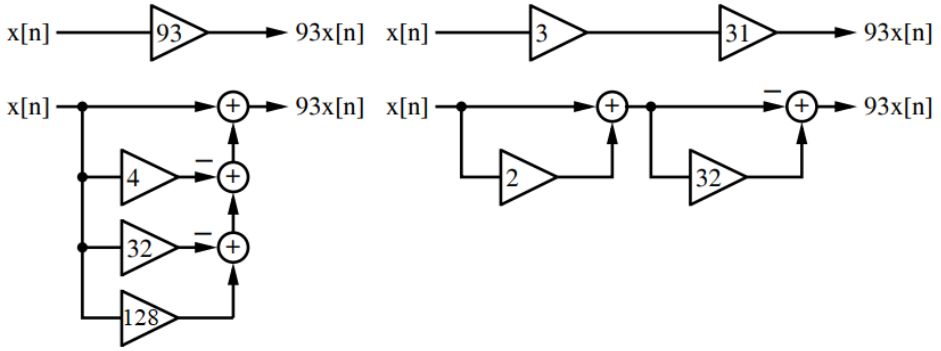

Figure 12: Dos realizaciones del producto $x \times 93_{10}$.

En general, la cantidad de sumas/restas será denominado como el costo de la multiplicación por la constante c. Hay muchas formas de combinar los factores de forma tal de obtener el mismo costo. Sea $k_i \in \mathbb{N}_{\geq 0}$, entonces:

- Costo 1: $c = 2^{k_0}(2^{k_1} \pm 2^{k_2})$
- Costo 2 (1): $c = 2^{k_0}(2^{k_1} \pm 2^{k_2} \pm 2^{k_3})$
- Costo 2 (2): $c = 2^{k_0}(2^{k_1} \pm 2^{k_2})(2^{k_3} \pm 2^{k_4})$
- Costo 3 (1): $c = 2^{k_0}(2^{k_1} \pm 2^{k_2} \pm 2^{k_3} \pm 2^{k_4})$
- Costo 3 (2): $c = 2^{k_0}\{[(2^{k_1} \pm 2^{k_2})2^{k_3} \pm 2^{k_4}]2^{k_5} \pm 2^{k_6}(2^{k_1} \pm 2^{k_2})\}$
- Costo 3 (3): $c = 2^{k_0}[(2^{k_1} \pm 2^{k_2})2^{k_3} \pm 2^{k_4}](2^{k_5} \pm 2^{k_6})$
- Costo 3 (4): $c = 2^{k_0}((2^{k_1} \pm 2^{k_2})(2^{k_3} \pm 2^{k_4}) \pm 2^{k_5})$
- Costo 3 (5): $c = 2^{k_0}(2^{k_1} \pm 2^{k_2})(2^{k_3} \pm 2^{k_4} \pm 2^{k_5})$
- Costo 3 (6): $c = 2^{k_0}(2^{k_1} \pm 2^{k_2})(2^{k_3} \pm 2^{k_4})(2^{k_5} \pm 2^{k_6})$
- Costo 3 (7): $c = 2^{k_0}[(2^{k_1} \pm 2^{k_2})2^{k_3} \pm (2^{k_4} \pm 2^{k_5})2^{k_6}]$

La Fig. 13 muestra todas las combinaciones hasta costo 4. Esto se conoce como multiplier-adder graph.

Una vez obtenido los MAG de cada coeficiente $h[k]$ del filtro, se buscan los subgrafos comunes entre los distintos coeficientes para poder reutilizarlos. En general se emplea un método heurístico denominado Reduced adder graph (RAG). Consta de los siguientes pasos:

1. Tomar el valor absoluto de todos los coeficientes ya que si alguno es negativo, esto puede compensarse en el adder block.
2. Hacer todos los coeficientes impares multiplicando por algún 2^{-k_i}, $(k_i \in \mathbb{N}_{\geq 0})$ ya que esto puede hacerse mediate conexionado.
3. Implementar todos los grafos de costo 1.
4. Usar los coeficientes de costo 1 en implementar los coeficientes de mayor costo.

La desventaja de este enfoque es que los grafos pueden ser muy profundos implicando mucha candidad de sumadores en cascada (logic depth).

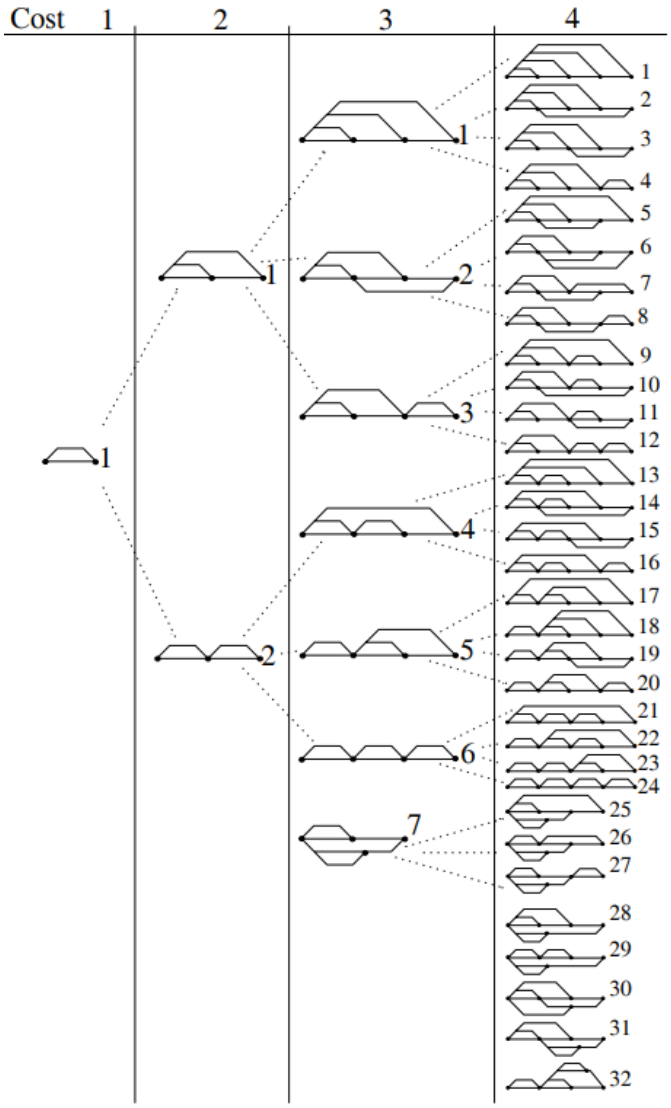

Figure 13: Todas las posibles opciones de como construir el grafo de sumadores para costos de 1 a 4.

6.5 Hartley's method

El método de Hartley [6] intenta reducir la profundidad lógica de las subexpresiones. El método parte de codificar todos los $h[k]$ en CSD de manera de minimizar la cantidad de no-'0' dígitos en cada coeficiente. A partir de esta codificación se arma una tabla de todos los $h[k]$ para luego buscar repeticiones de subexpresiones.

Ejemplo:

k	$h[k]$
0	$0001101_{2C} = 0010\bar{1}01_{CSD}$ $(+13_{10})$
1	$0010110_{2C} = 010\bar{1}0\bar{1}0_{CSD}$ $(+22_{10})$
2	$1001001_{2C} = 1001001_{CSD}$ $(+73_{10})$
3	$0111011_{2C} = 1000\bar{1}0\bar{1}_{CSD}$ $(+59_{10})$
4	$0101110_{2C} = 10\bar{1}00\bar{1}0_{CSD}$ $(+46_{10})$
5	$0111101_{2C} = 1000\bar{1}01_{CSD}$ $(+61_{10})$
6	$0100011_{2C} = 0100\bar{1}0\bar{1}_{CSD}$ $(+35_{10})$
7	$0011000_{2C} = 010\bar{1}000_{CSD}$ $(+24_{10})$

Se arma una tabla de los coeficientes en CSD sin los ceros para facilitar la lectura. Se busca el patrones más

comunes y se extraen de la tabla y así se continua hasta que no haya más patrones para extraer. Observar que $1\ldots\overline{1}$ es equivalente a $\overline{1}\ldots1$ y también $1\ldots1$ es equivalente a $\overline{1}\ldots\overline{1}$.

Observar que muchas diferentes subexpresiones se repiten la misma cantidad de veces. Por lo cual es arbitrario cual de las subexpresiones de mayor repetición se toma como subexpresión a eliminar. Algún criterio puede usarse como ser tener preferencia por una subexpresión horizontal o vertical y luego continuar por las oblicuas, etc.

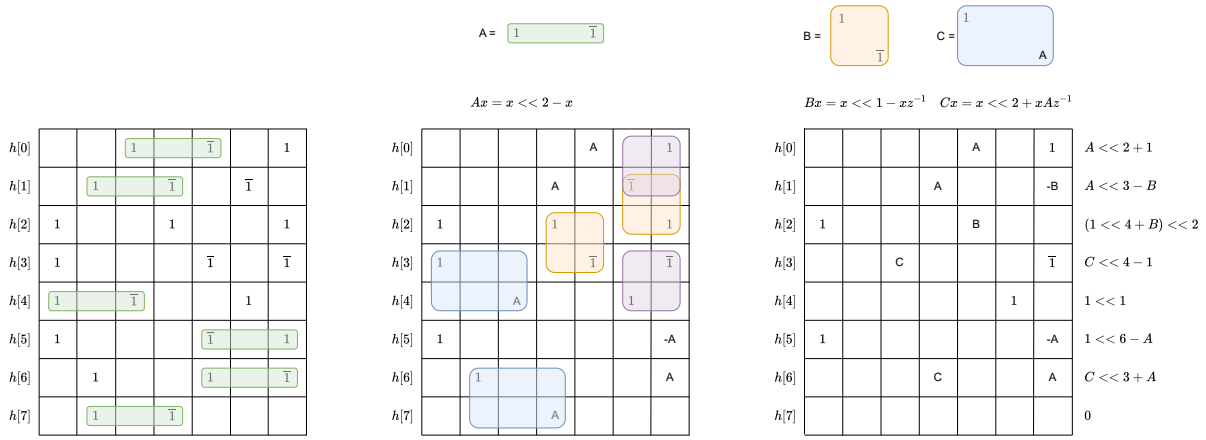

Figure 14: Procedimiento de CSE de Hartley.

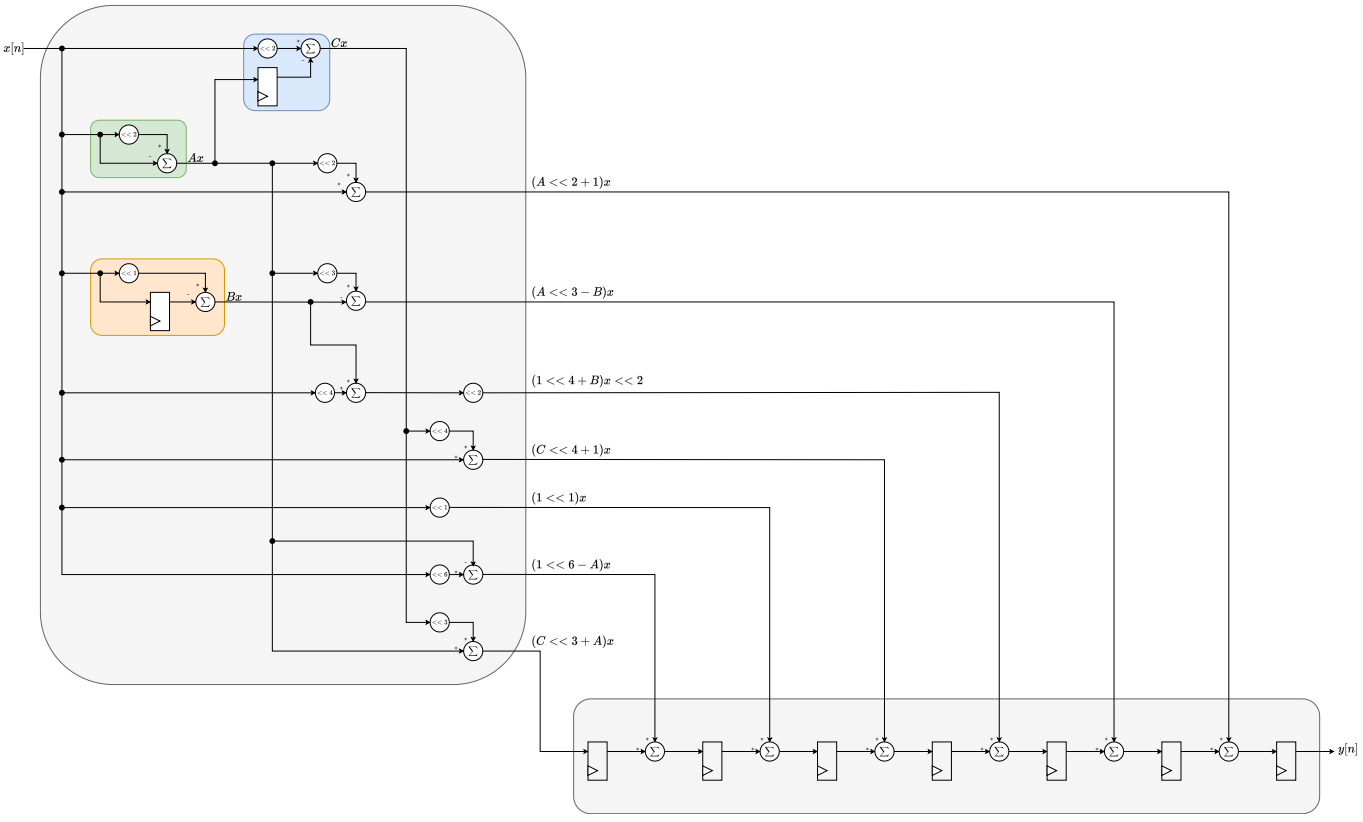

Figure 15: Filtro obtenido mediante el procedimiento de Hartley.

Diferentes métodos existen para mejorar aún el procedimiento de Hartley. Por ejemplo, en lugar de codificar los coeficientes en CSD, estos se podrían codificar en MSD (Minimum Signed Digit). Por ejemplo, $12_{10} = 10\overline{1}00_{CSD} = \{10\overline{1}00, 1100\}_{MSD}$. OBservar que en MSD, el número 12_{10} admite dos representaciones que respetan la condición de mínima cantidad de no-'0' dígitos, pero flexibiliza la condición de que dos dígitos consecutivos, al menos uno debe ser '0'. Mediante la codificaciόm MSD, aplicando el método de Hatley se podrían obtener distintas implementaciones del mismo filtro y elegir de todas ellas las que tenga más subexpressiones a eliminar. Se deja como referencia algunos trabajos al respecto.

7 Tamaño de palabra

Observemos que en 2C, si $x[n]$ posee B bits de palabra, entonces $x_{\max} = 2^{B-1} - 1$ y $x_{\min} = -2^{B-1}$. Por lo cual se sabe de la ec. (6) que el valor máximo de la salida está dado por

$$|y[n]| \leq 2^{B-1} \sum_{k=0}^{L-1} |h[k]| \tag{40}$$

Por lo cual, recordando el teorema de wrap-around de 2C, se sabe que si todas las sumas parciales son calculados con $B + \left\lceil \log_2 \left(\sum_{k=0}^{L-1} |h[k]| \right) \right\rceil$ bits entonces debe ser suficiente para que el valor de $y[n]$ sea correctamente representado. Sin embargo, notemos que cada producto parcial $x[n-k] \times h[k]$ es suficiente con ser representado con $B + \lceil \log_2(h[k]) \rceil$ bits. Por otro lado, cada suma parcial definida por

$$s_i = \sum_{k=0}^{i} x[n-k]h[k] \tag{41}$$

es suficiente con que sea representada por $B + \left\lceil \log_2 \left(\sum_{k=0}^{i} |h[k]| \right) \right\rceil$. Con lo cual, en conclusión, se puede asegurar que si cada producto parcial y suma parcial es representado por los siguientes números de bits entonces el filtro no puede sufrir de overflow:

- Número de bits de un producto parcial: $B + \lceil \log_2(h[k]) \rceil$
- Número de bits de una suma parcial: $\min B + \left\lceil \log_2 \left(\sum_{k=0}^{i} |h[k]| \right) \right\rceil, B + \left\lceil \log_2 \left(\sum_{k=0}^{L-1} |h[k]| \right) \right\rceil$
- Número de bits de la salida: $B + \left\lceil \log_2 \left(\sum_{k=0}^{i} |h[k]| \right) \right\rceil$

Si por el contrario, se desea una cantidad menor de bits, entonces saturación más redondeo/truncado debe ser implementado en los cálculos de las sumas y los productos parciales.

References

[1] S. A. White, "Applications of distributed arithmetic to digital signal processing: a tutorial review," in IEEE ASSP Magazine, vol. 6, no. 3, pp. 4-19, July 1989.

[2] G. NagaJyothi and S. SriDevi, "Distributed arithmetic architectures for FIR filters-A comparative review," 2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET), Chennai, India, 2017, pp. 2684-2690.

[3] D. R. Bull, D. H. Horrocks, "Primitive operator digital filters", IEE Proc. G. Vol. 138, No. 3, pp. 401-412, June 1991.

[4] A. G. Dempster and M. D. Macleod, "Use of minimum-adder multiplier blocks in FIR digital filters," in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 42, no. 9, pp. 569-577, Sept. 1995.

[5] M. Potkonjak, M. B. Srivastava and A. P. Chandrakasan, "Multiple constant multiplications: efficient and versatile framework and algorithms for exploring common subexpression elimination," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 15, no. 2, pp. 151-165, Feb. 1996.

[6] R. Hartley, "Subexpression Sharing in Filters Using Canonic Signed Digit Multipliers". IEE Transactions on Circuits and Systems - II: Analog and Digital Signal Processing, Vol. 43, No 10, Oct 1996.

[7] R. Pasko, P. Schaumont, V. Derudder, S. Vernalde and D. Durackova, "A new algorithm for elimination of common subexpressions," in IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 18, no. 1, pp. 58-68, Jan. 1999.

[8] In-Cheol Park and Hyeong-Ju Kang, "Digital filter synthesis based on minimal signed digit representation," Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232), Las Vegas, NV, USA, 2001, pp. 468-473.

[9] Hyeong-Ju Kang and In-Cheol Park, "FIR filter synthesis algorithms for minimizing the delay and the number of adders," in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 48, no. 8, pp. 770-777, Aug. 2001.

[10] M. Martinez-Peiro, E. I. Boemo and L. Wanhammar, "Design of high-speed multiplierless filters using a nonrecursive signed common subexpression algorithm," in IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing, vol. 49, no. 3, pp. 196-203, March 2002.