2 FIR Filter: Design and Structures

2.1 Linear Phase Response

The property to have an exactly linear phase response φ(ω) is one of the most important characteristics of FIR filters.

- **A filter with nonlinear phase response will cause distortions, because the frequency components in the signal will be delayed by an amount not proportional to frequency.**
- **As a result the harmonic relationship is altered.**

- **The group delay $\tau_{gr} = -d\varphi(\omega)/d\omega = \text{const.}$ of the filter provides a useful measure how the filter modifies the phase characteristics of input signals.**

- **The group delay is the average time delay the superposed signal components suffer at each frequency. The envelope of the input signal will be delayed but the symmetry of the signals shape is preserved only higher frequency components will be attenuated i.e. in case of a lowpass filter.**

- **Linear phase is supported under certain symmetry conditions which have to be fulfilled by the coefficients $h(n)$ of a filter with Nth order and $L = N + 1$ coefficients:**

Positive symmetry: $h(n) = h(N-n)$ **with** $n = 0, 1, \ldots, N/2$ **if N is even** $n = 0, 1, \ldots, (N-1)/2$ **if N is odd**	**Negative symmetry:** $h(n) = -h(N-n)$ **with** $n = 0, 1, \ldots, N/2$ **if N is even** $n = 0, 1, \ldots, (N-1)/2$ **if N is odd**
Linear Phase: $\varphi(\omega) = - a\omega$ **with** $a = N/2$	**Linear Phase:** $\varphi(\omega) = b - a\omega$ **with** $a = N/2$ **and** $b = \pi/2$ (90°)

Table 2-1: Symmetry conditions for filter coefficients in order to provide constant group delay.

Impulse response symmetry	Filter order N	Frequency response H(ω)	Filter type
Positive	**Even**	$e^{-j\omega TN/2}\sum_{k=0}^{N/2} a(k)\cos(\omega Tk)$	**1**
	Odd	$e^{-j\omega TN/2}\sum_{k=1}^{(N+1)/2} b(k)\cos(\omega T(k-1/2))$	**2** **no highpass**
Negative	**Even**	$e^{-j(\omega TN/2-\pi/2)}\sum_{k=1}^{N/2} a(k)\sin(\omega Tk)$	**3** **no lowpass,** **no highpass**
	Odd	$e^{-j(\omega TN/2-\pi/2)}\sum_{k=1}^{(N+1)/2} b(k)\sin(\omega T(k-1/2))$	**4** **no lowpass**
a(0) = h(N/2); a(k) = 2h(N/2-k); b(k) =2h((N+1)/2-k); ω = 2πf; T = 1/f$_S$			

Table 2-2: Four types of FIR filters with linear phase

- The comments on the filter types which can be realised and which can not follows a check of the frequency response at two frequency points:
  - $\omega = 0$: DC gain is given by sum of the coefficients $\Sigma h(k)$.

    With negative symmetry of coefficients this sum is always zero and therefore no lowpass can be realised with type 3 and 4.

    DC gain be also evaluated from the z-transfer function by applying the final value limit formular.

    $\lim_{z \to 1} (z-1)Y(z)$ with the step response $Y(z) = H(z)z/(z-1)$

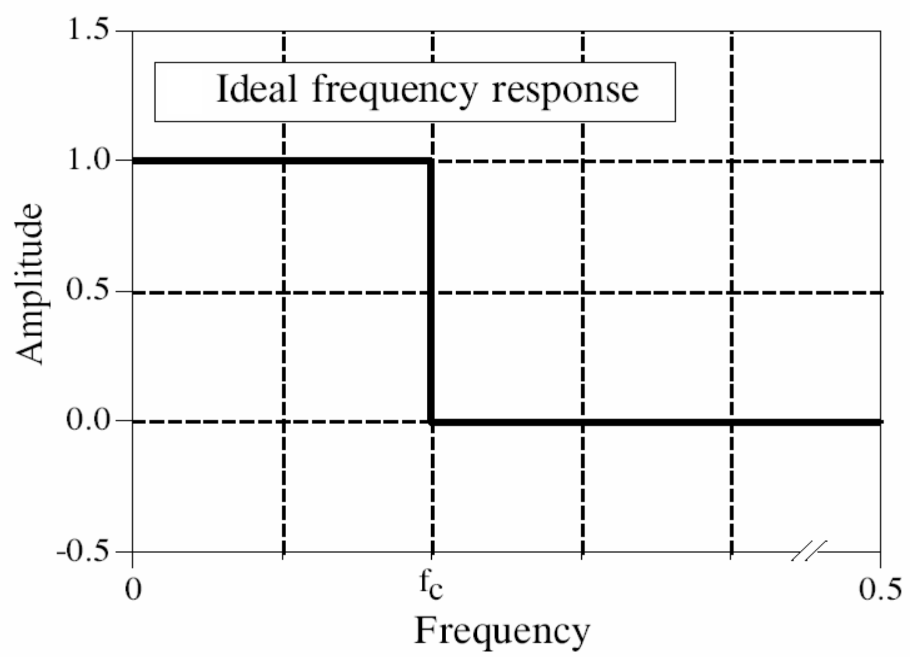
  - $\omega = \omega_N = \omega_S/2$: At the right border of the allowed frequency range (Nyquist frequency) only highpass filters offer a gain unequal zero.

    With filter type 2 and 3 the terms $\cos(T\omega_S/2(n-1/2))$ and $\sin(nT\omega_S/2)$ respectively will always be zero and therefore no highpass characteristic is possible.

- Type 1 is the most suitable of the four. When specifying a frequency response and the calculation results suggest an odd filter order with positive symmetry then we will just increment the order N by one and all filter characteristics will be available.

2.2 FIR Design with Window Method

- **This method makes use of the fact that FIR filter coefficients and the impulse responses of this non recursive filter type are directly related.**
- **Therefore the starting point of the filter specification could be the desired impulse response. Sampling the response will directly yield the needed coefficients for implementation in i.e. taped delay chain as shown in chapter 1.3.**



- **But usually it is more handy to start with a given frequency response which is specified by a cutoff frequency f_C in order to separate pass band from stop band.**
- **Coefficient calculation is derived from the ideal lowpass response $H_D(\omega)$ (D desired) which is determined by a rectangular shape.**

Fig. 2-1: Frequency response $H_D(\omega)$ of the ideal lowpass[2].

- **The impulse response h(k) is related to the inverse Fourier transform of H(ω). Here the pair of transformation is used which couples discrete time domain signals (sequence of samples) with a continuous spectral description:**

Discrete Time Fourier Transformation

$$H(\omega) = \sum_{k=-\infty}^{\infty} h(k) e^{-j2\pi fTk} \qquad 2.1$$

Inverse Discrete Time Fourier Transformation

$$h(k) = T \int_{-fs/2}^{fs/2} H(\omega) e^{j2\pi fTk} \, df \qquad 2.2$$

- **The evaluation of integral 2.2 has to be applied on Fig. 2-1 within the frequency range –$f_C \le f \le f_C$ where H(ω) is one. Usually the integrating variable f is substituted by a normalised frequency $\Omega = 2\pi fT$.**

Impulse response:

$$h(k) = \frac{1}{2\pi} \int_{-\Omega c}^{\Omega c} H(\Omega) e^{j\Omega k} \, d\Omega \qquad 2.3$$

- **With substitution by Euler's expression the complex part in 2.3 will have no influence on the final result which is the sinc-function:**

$$h(k) = \sin(2\pi f_C Tk)/\pi k \qquad ; k \neq 0, -э \leq k \leq э \qquad 2.4$$

$$h(0) = 2f_C T \qquad ; k = 0 \text{ using L'Hospital's rule}$$

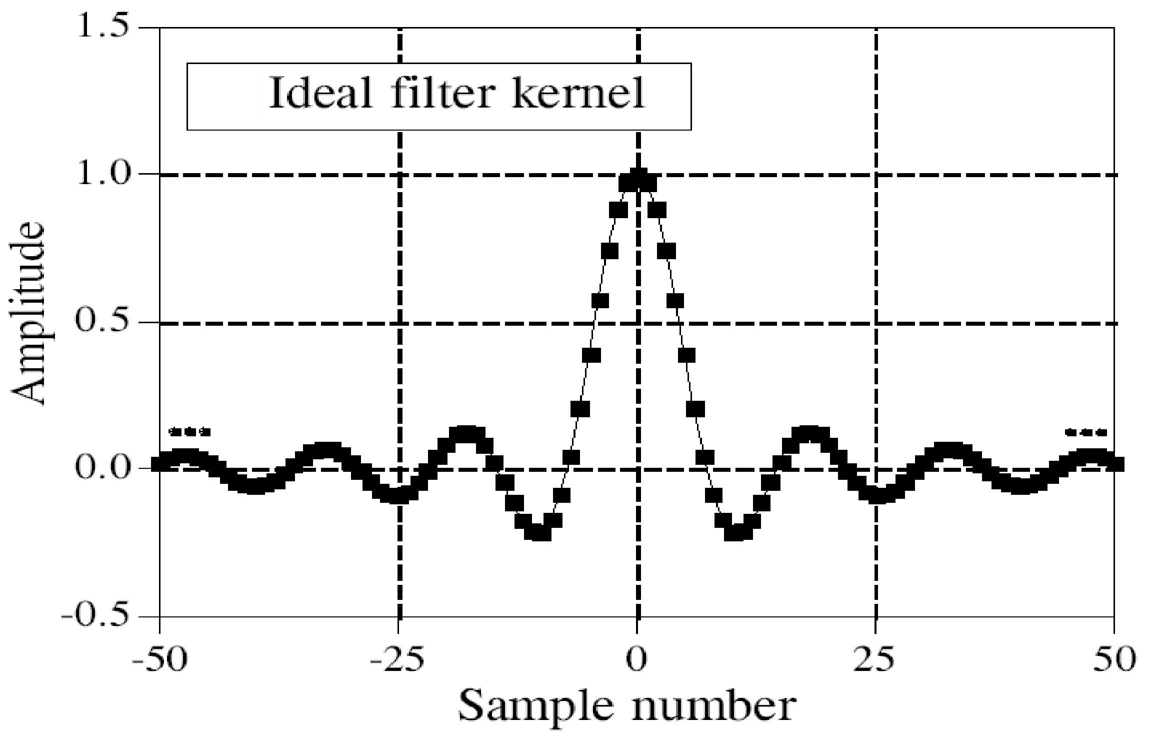


Fig. 2-2: Impulse response $H_D(\omega)$ of the ideal lowpass filter with scaled amplitude h(k)/h(0) [2].

- **Two problems are related with this result:**

➢ **The infinite length of the impulse response becomes a show stopper for computer application.**

Namely we evaluate a filter output y(k) by convolving this filter kernel h(k) with an input signal sequence x(k). Therefore the convolution h(k)*x(k) will have an infinite duration which is not realistic.

The result is simply truncating the impulse response 2.4 on both sides symmetrically which provides a set of samples with N+1 elements: i.e. N = 100 with N/2 = 50 on both sides in Fig. 2-2.

This windowing procedure makes a finite convolution available.

➢ **From Fig. 2-2 we recognise a non causal time domain representation which offers an impulse response which starts earlier than the input signal has an impact on the system.**

Therefore it is necessary to apply a right shift in time with an TN/2 offset. This time shift of the time limited impulse response introduces a negative phase to the frequency response $H(\omega)e^{-j\omega kT}$.

- **With the following pages Matlab code is provided which supports evaluation of h(k) sinc function and corresponding frequency response calculations according to the expressions presented in the introduction.**

```
% 1. conditions:
fc = 12000; % cutoff frequency in Hz.
fs = 48000; % sample frequency in Hz.
N = 20; % filter order: limited number of samples
T = 1/fs; % sample period
wc = 2*pi*fc; % rotational (angular) frequency
% 2. sampled impulse response:
k = linspace((-N/2),(N/2),(N+1)); % limited number of samples!
% sample index k, symmetrically around 0
h_k = (sin(wc*k*T)./(pi*k)); % samples: array operation
h_k((N/2)+1) = 2*fc/fs ; % corrected value at k=0
figure(1); % a new window for each plot
stem(k,h_k); %  samples output: single values
grid;
title('Sampled Impulse Response');
xlabel('Sample Number');
ylabel('Amplitude');
```

Code 2-1: Sampled non causal impulse response. N = 20, f_S = 48kHz, f_C = 12kHz

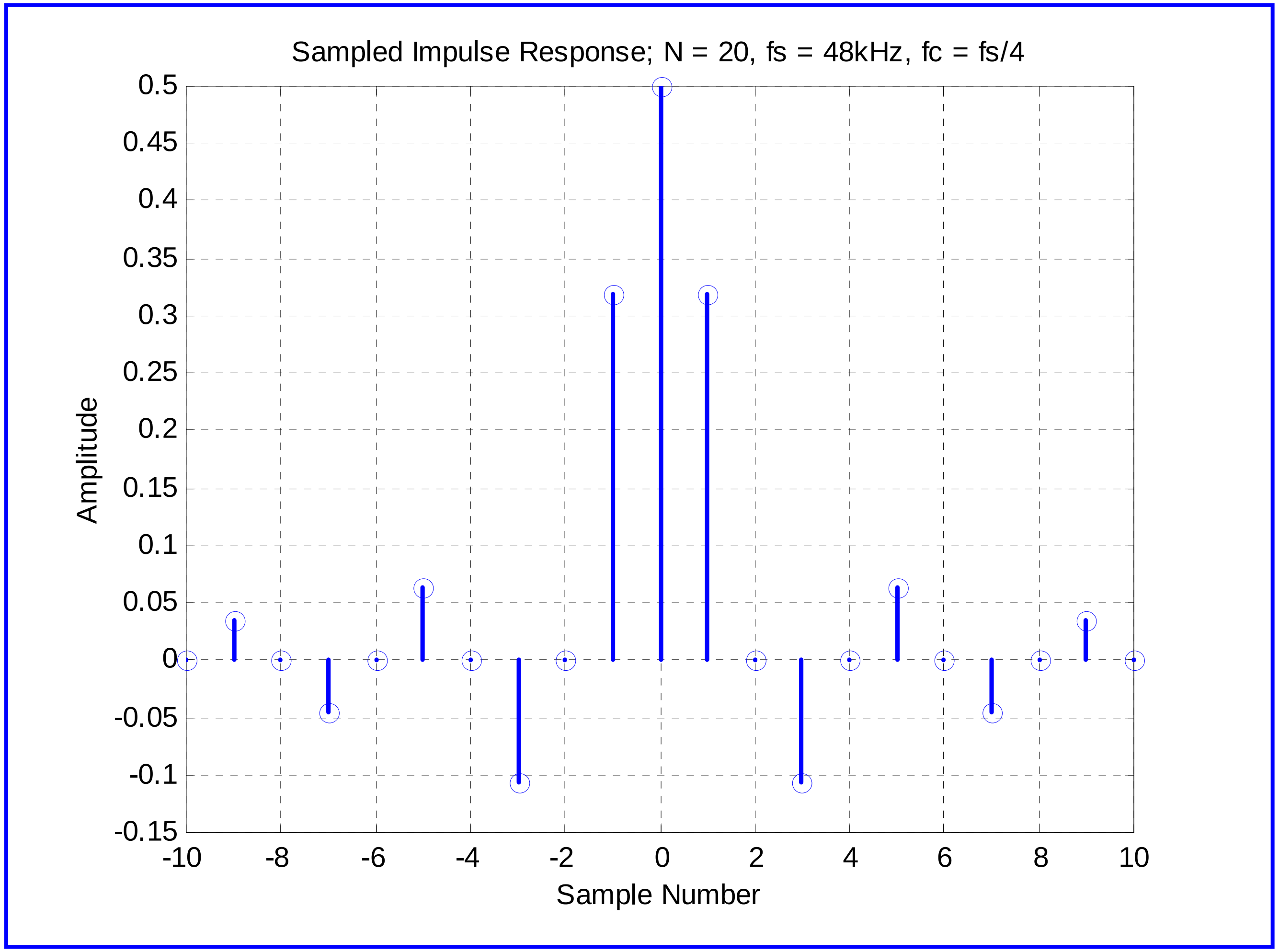


Fig. 2-3: Non causal impulse response (rectangular window).

- With this strategy for lowpass design a positive symmetry of filter coefficients is directly available.
- Because of chosen cutoff frequency $f_C=f_S/4$ here a special result is given with h(k) = 0 for all even k values.
- The sum of the samples is not correctly equal to 1 which has been expected because of Fig. 2-1. The more samples will be used with increasing N the dc gain tends exactly to become 1.

- The frequency response for the set of limited coefficients h(k) is calculated by the function freqz. [matlab12\help\toolbox\signal\freqz.html].

```
figure(2);
%3.frequency response evaluated with H(z) nominator and denominator
freqz(h_k,1,512,48000); % direct plot of magnitude and phase
sum = 0;% dc gain
for i=1:N+1
   sum = sum +h_k(i);
end;
```

Code 2-2: A complex frequency response H(ω) given by H(z) evaluated at $z = e^{j\omega t}$.

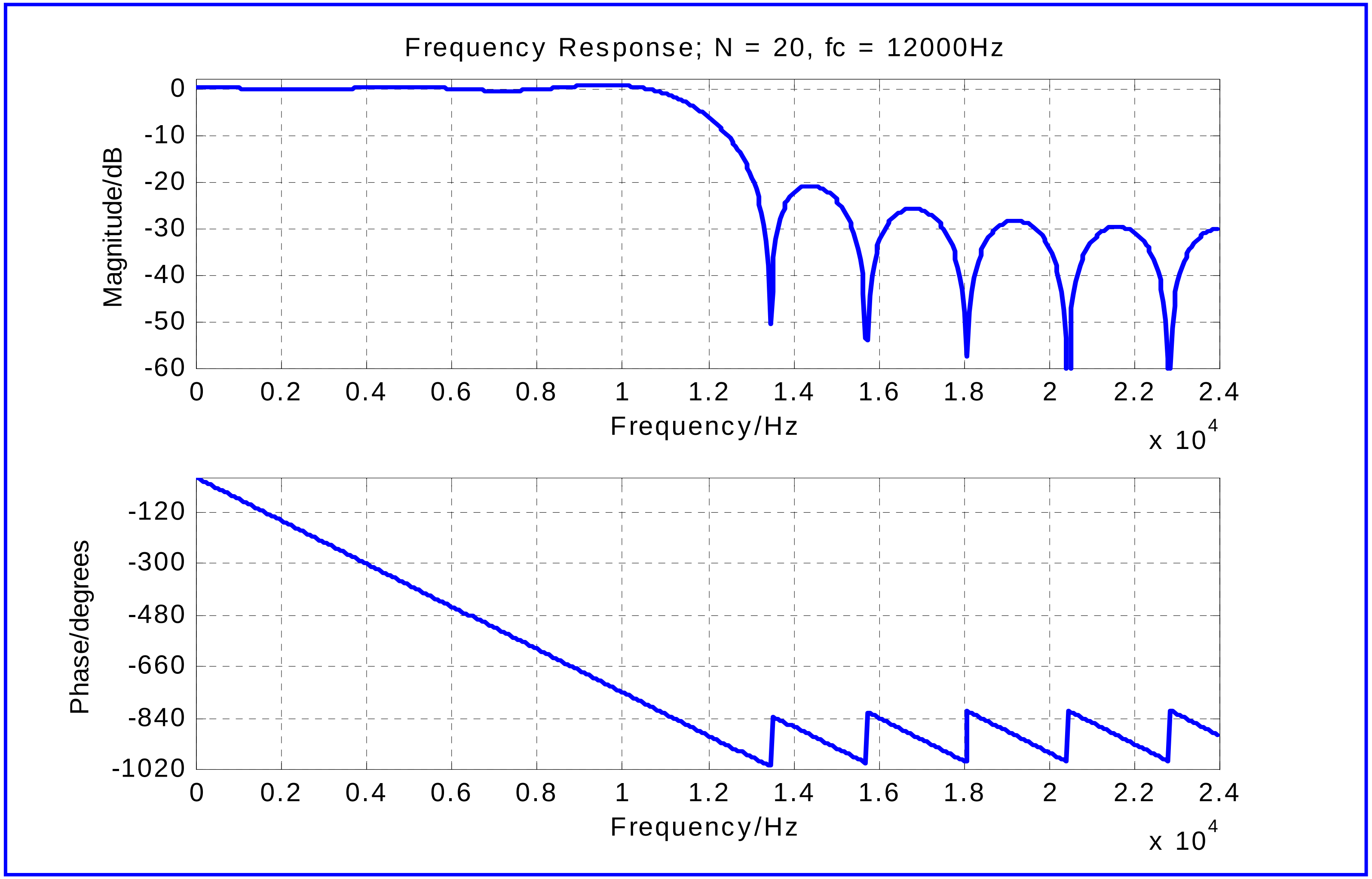


Fig. 2-4: Frequency response of FIR lowpass with order N = 20, rectangular window, f_S = 48kHz.

- With the truncation of the impulse response by a rectangular window w(k) in the time domain most of the coefficients of an ideal lowpass will be discarded. This time domain manipulation introduces several effects to the shape of the frequency response in Fig. 2-4:
  - ➢ Sidelobes with non zero attenuation arise because of needed higher frequency components than f_C which model the abrupt transition to zero of the limited impulse response.
  - ➢ A passband ripple and gain overshoots are called Gibb's phenomenon.

- Both effects can be explained with the windowing which is a multiplication in time domain:
  - ➢ In the frequency domain this is equivalent to convolving $H_D(\omega)$ (Fig. 2-1) with $W(\omega)$, where $W(\omega)$ is the discrete time Fourier transform of w(k).
  - ➢ As $W(\omega)$ has the typical sinx/x shape with amplitude oscillation the convolution process causes passband and stopband ripples in the result $H(\omega) = H_D(\omega) * W(\omega)$
  - ➢ As the $W(\omega)$ kernel is shifted to the right as principally shown in chapter 1.3 the sum of positive and negative values of $W(\omega)$ under the rectangle $H_D(\omega)$ oscillates.
  - ➢ When the mainlobe of $W(\omega)$ leaves the rectangle $H_D(\omega)$ there will be ripples in $H(\omega)$ beyond the cutoff frequency.
  - ➢ At least the ripples in $H(\omega)$ are caused by the sidelobes of $W(\omega)$.

- How many sinx/x coefficients have to be used, or how wide must w(k) be, to get no ripples in H(ω) and to get sharp edges in H(ω)?
    - As long as w(k) is a finite number of unity values there will be sidelobe ripples in W(ω) and this will induce passband ripples and sidelobes in H(ω) frequency response.
    - As long there is a instantaneous discontinuity in the window function w(k) an infinite set of frequency components is needed to represent w(k) by a Fourier series which gives another reason for sidelobes.
    - More coefficients can be used by extending the width of the rectangular w(k) to reduce the filter transition region, but a wider w(k) does not eliminate the filter passband ripple, as long as w(k) has sudden discontinuities.

- An alternative approach for magnitude |H(ω)| calculation is suggested with Code 2-3 [matlab12\help\techdoc\ref\polyval.html].
    - The magnitude in Fig. 2-5 is evaluated up to $f = f_S$ so the mirrored shape at f_N becomes apparent.
    - Characteristics because of rectangular window: Equal passband and stopband ripple of 0.7416 dB (0.0891) and a transition width of $\Delta f = 0.9f_S/(N+1) = 2.06$kHz.

```
% 4. direct analysis of H(w) with substituted z variable
fHz = linspace(0,fs,(fs/10)+1);
% frequency axis 0<f<fs, increment with delta = 10Hz
fHz0 = (2*fHz)./fs; % scaled frequency: f/(fs/2) Nyquist freq.
omega = 2*pi*fHz; %Substitution of z: evaluation along the z-unit circle
z = exp(sqrt(-1) * omega / fs);
hw = abs(polyval(h_k,z)); % magnitude |H(w)|
hw0 = 20*log10(hw);
figure(3);
plot(fHz,hw); %magnitude with passband ripple:limited number of samples
grid; title('Frequency Response');
xlabel('Frequency f/Hz');
ylabel('Magnitude');
axis([0 fs 0.0 1.1]);% f-range mag-range
figure(4);
plot(fHz0,hw0);% not depicted
grid;
title('Frequency Response');
xlabel('Frequency scaled f/(fs/2)');
ylabel('Magnitude/dB');
axis([0.0 1.0 min(hw0)+ 20 (max(hw0)+2)]);% f < fs/2
```

Code 2-3: Magnitude |H(ω)| calculation based on polynomial H(z) with substituted z.

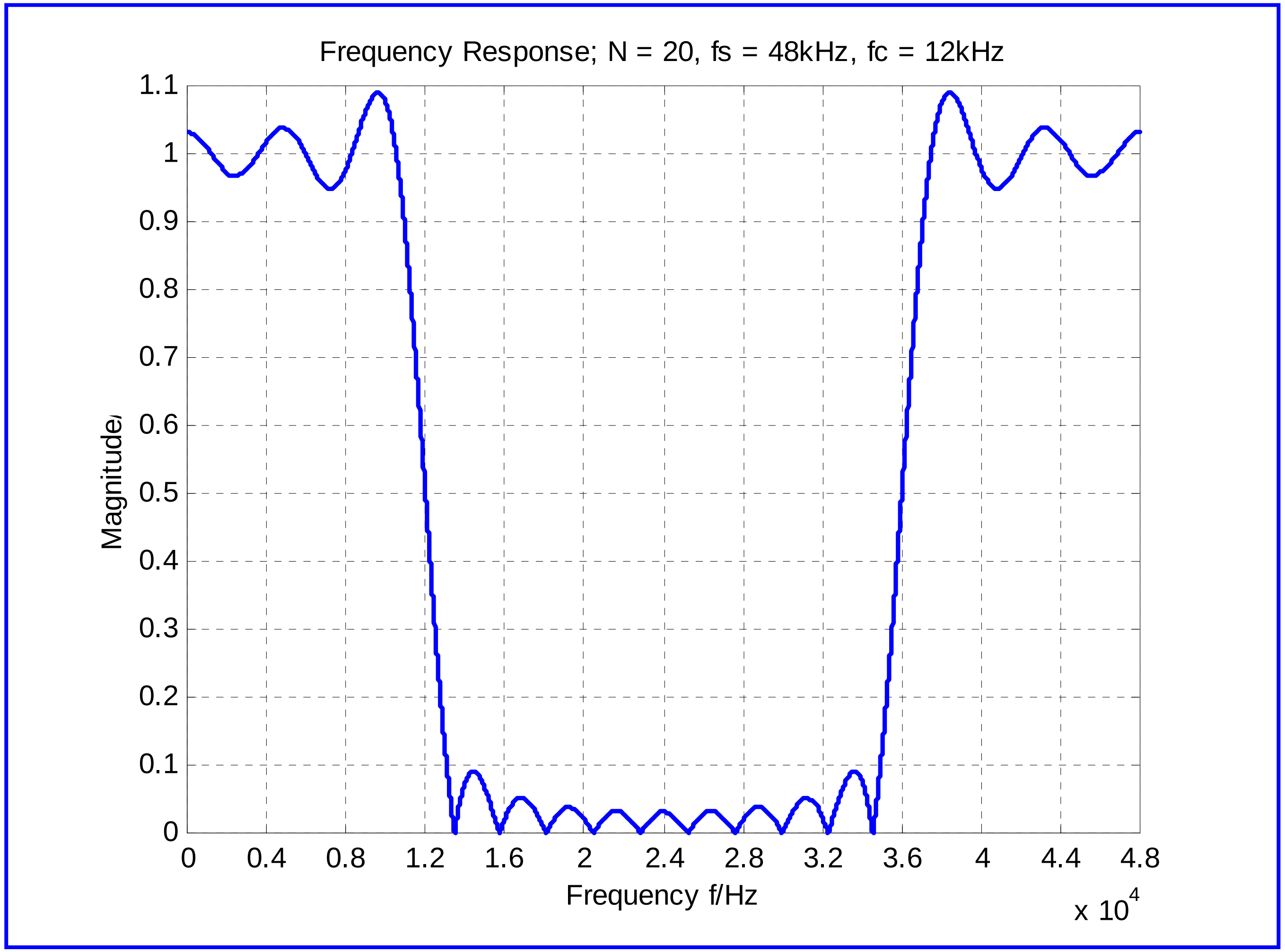


Fig. 2-5 : FIR lowpass frequency response. |H(ω)| is periodically with f_S.

- The only choice for a method to reduce passband and stopband ripple which means increased sidelobe attenuation is to avoid the instantaneous discontinuities with windowing.

- Window w(k) functions which decay smoothly towards zero have to be chosen.

- One of the most widely used window functions is the Hamming window which is defined as:

$$w(k) = 0.54 - 0.46\cos(2\pi k/N) \quad 0 \leq k \leq N \qquad 2.5$$

- ➢ The Matlab code Code 2-4 describes Hamming window calculation and coefficient weighting (comp. Fig. 2-6).
- ➢ The corresponding frequency domain representation of Fig. 2-6 has a wider mainlobe than the rectangular window, but sidelobes are smaller relative to the mainlobe (not depicted).
- ➢ As a result the transition width of a filter designed with the Hamming window is larger $\Delta f = 3.3 f_S/(N+1) = 7.54$kHz with N = 20, but the maximum stopband attenuation is increased up to about 53 dB. The maximum peak passband ripple is about 0.0194 dB

```
% 4. Window functions
n1 = linspace(0,N,(N+1)); % positive index values for causal FIR filter
w2 = 0.54-(0.46*cos((2*pi*n1)/N));% Hamming window
figure(5);
stem(n1,w2);
grid;
title('Hamming window');
xlabel('Sample number');
ylabel('Value');
% 5. Weighted coefficients
h_k2 = h_k .* w2;% each coefficient multiplied with according weight
h_k2((N/2)+1) = (2*fc/fs) * w2((N/2)+1);%center coefficient substituted
% 6. Plots of weighted coefficients
figure(6);
stem(n1,h_k2);
grid;
title('FIR filter coefficients with Hamm window');
xlabel('Number');
ylabel('Value');
axis([(0) (N) -.2 .5]);
```

Code 2-4: Hamming window calculation and weighting of impulse response; N = 20.

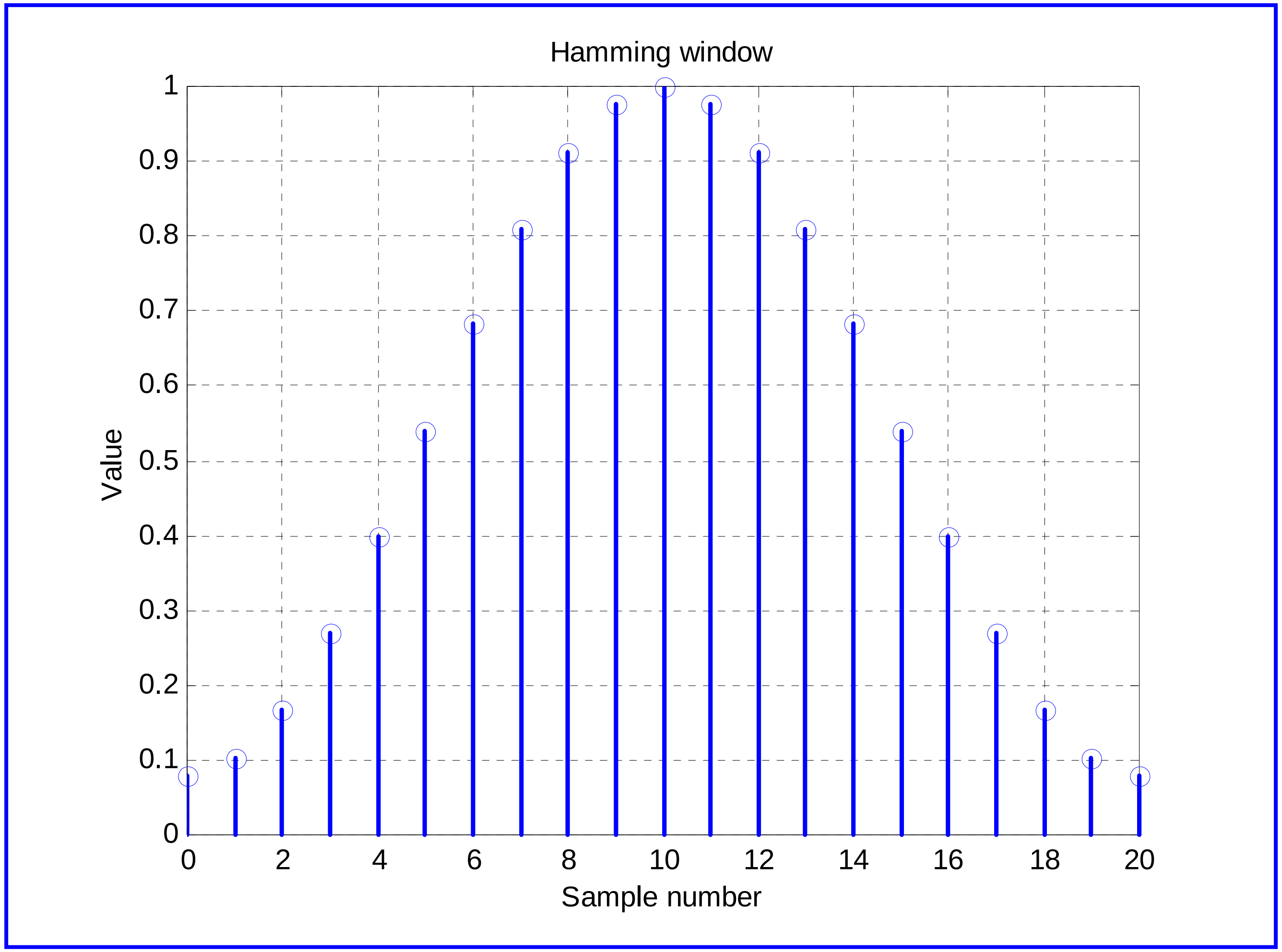


Fig. 2-6: Time domain characteristic of a Hamming window; N = 20.

```
sum2 = 0;% dc gain
for i=1:N+1
   sum2 = sum2 +h_k2(i);
end;
% 7. frequency response directly
figure(7);
freqz(h_k2,1);
% Evaluation of |H(w)|
hw02 = abs(polyval(h_k2,z)); %  Hamming
hw2 = 20*log10(hw02);
figure(8);
plot(fHz0,hw2); %
grid;
title('Frequency Response with Hamm window');
xlabel('Frequency scaled f/(fs/2)');
ylabel('Magnitude/dB');
axis([0.0 1.0 -80 (max(hw2)+2)]);
```

Code 2-5 : Frequency response calculation with weighted coefficients by Hamming window.

- The larger transition width of the lowpass in Fig. 2-7 is caused by the wider mainlobe of the Hamming window’s frequency domain representation. The sidelobe attenuation is more than –30 dB better than in Fig. 2-4.

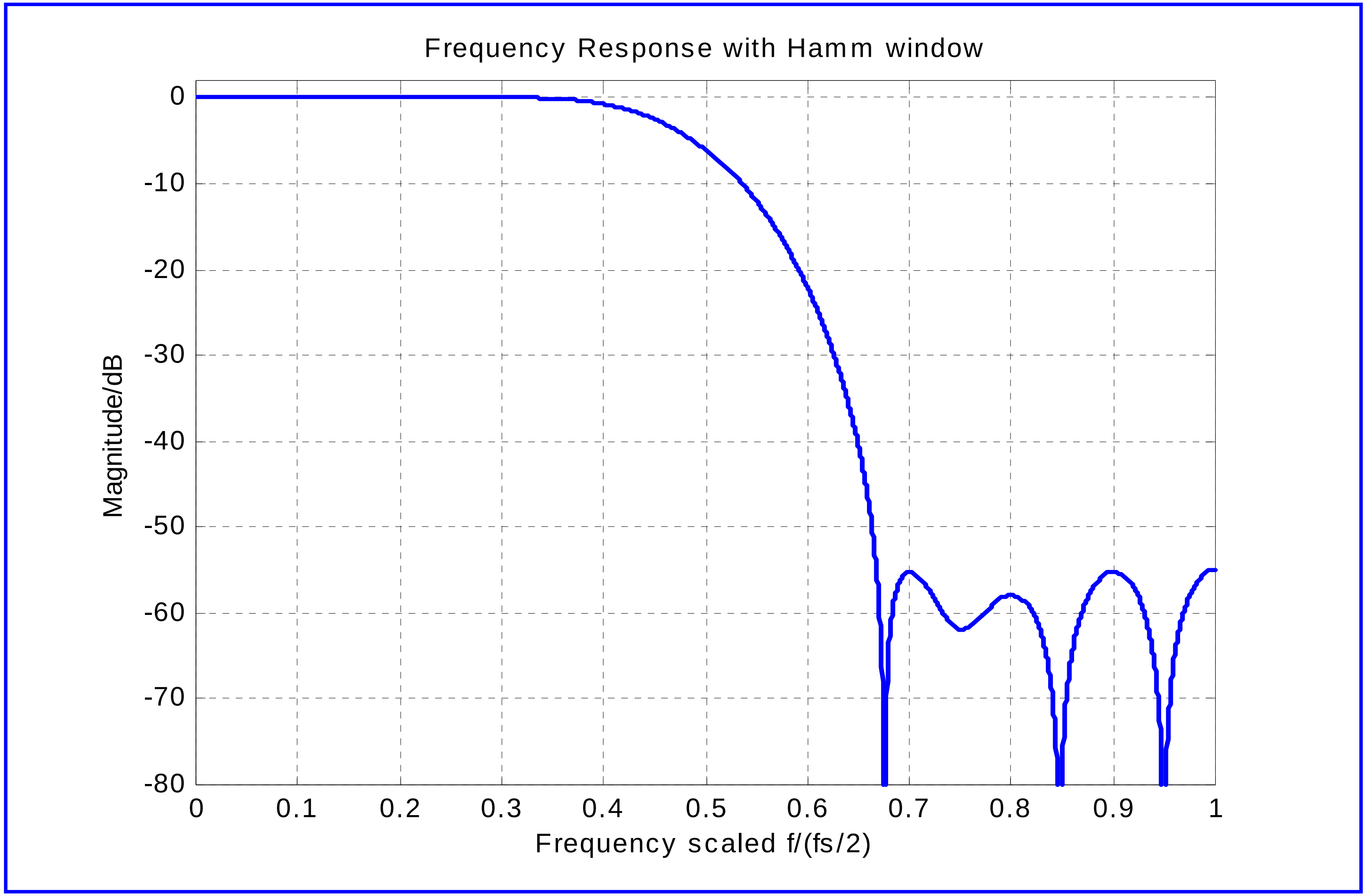
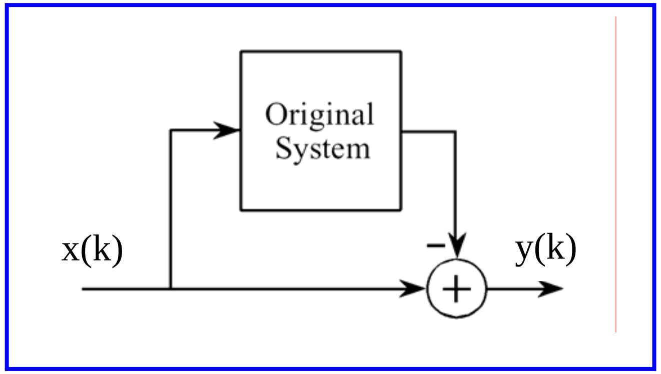


Fig. 2-7: Lowpass filter designed with Hamming window; N = 20, f_S = 48kHz, f_C = 12kHz.

- All other typical filter types like highpass, bandpass and bandreject can be derived from a low-pass design by applying one of the following two methods to a highpass [2, chapt. 14]:
  - ➢ Spectral inversion: Change passband into stopband and vice versa.
  - ➢ Spectral reversal: Shift of H(ω) along the frequency axis by $f_S/2= f_N$ to the right.

- Spectral inversion is performed by subtracting the output of the filter from the original signal:



- ➢ $H_{hp}(\omega) = 1 – H_{lp}(\omega)$
- ➢ According to filter type 1 in Table 2-2 this means:

  All weights a(k) of cos-terms will have a changed sign and for k = 0 it results in a(0) = 1- h(N/2).
- ➢ The cutoff frequency f_C will be unchanged.

- ➢ Related to the filter kernel h(k) two steps have to be done: change the sign of each coefficient and add 1 to the center value -h(N/2).

- Spectral reversal is based on the symmetry of the H(ω) shape which is periodically with f_S.
  - A shift of H(ω) with f_N to the right provides a highpass with a cutoff frequency $f_{Cnew} = f_N - f_C$.
  - The frequency shift applied to filter type 1 in Table 2-2 introduces terms like $\cos((\omega - \omega_S/2)Tk)$ which result in products $\cos(\omega Tk)\cos(\pi k)$. Therefore the weights will have alternating signs and the center value remains unchanged.
  - Related to the filter kernel h(k) the sign of every other coefficient has to be changed.

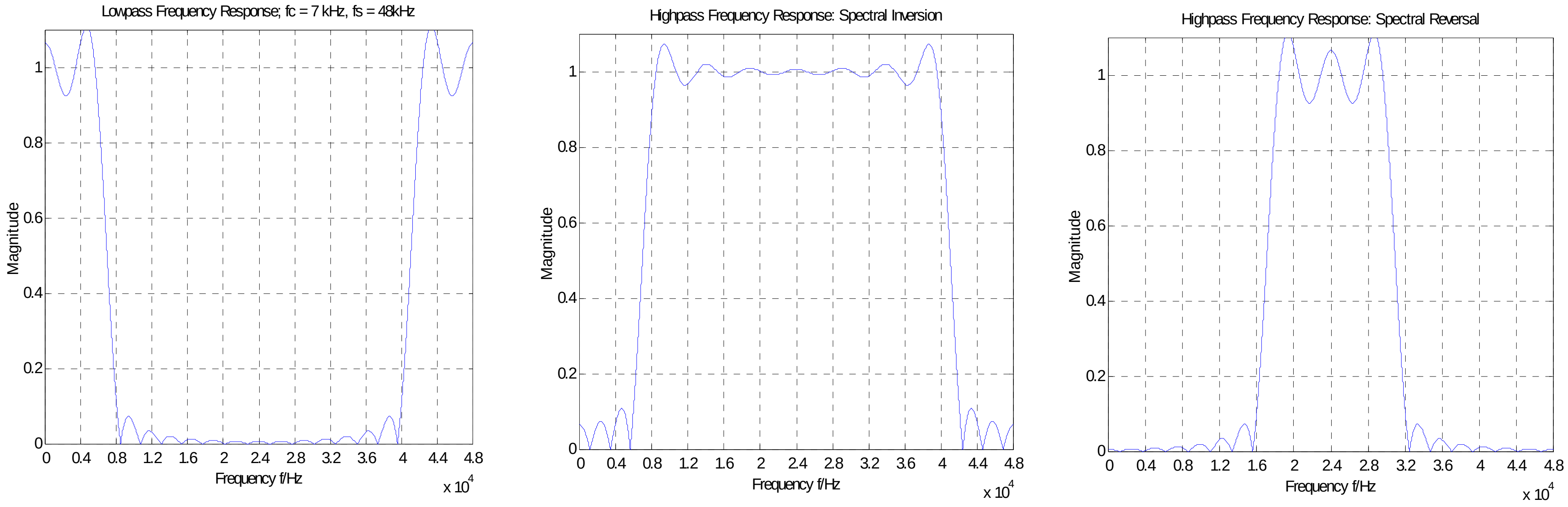


Fig. 2-8: Lowpass to highpass conversion with spectral inversion and reversal; N = 20, f_{Clp} = 7kHz, f_N = 24kHz.

2.3 Direct and Transposed Forms

2.3.1 Direct Form I

- **This structure is directly related to the difference equation.**
  - **A chain of N register stages provides all delayed input signals x(n –k).**
  - **N+1 multiplications are processed in parallel with coefficients $c_k = h(k)$. Products are added with one adder which will become an adder chain in FPGA hardware.**
  - **The longest signal path is build by the N adder chain and one multiplier.**

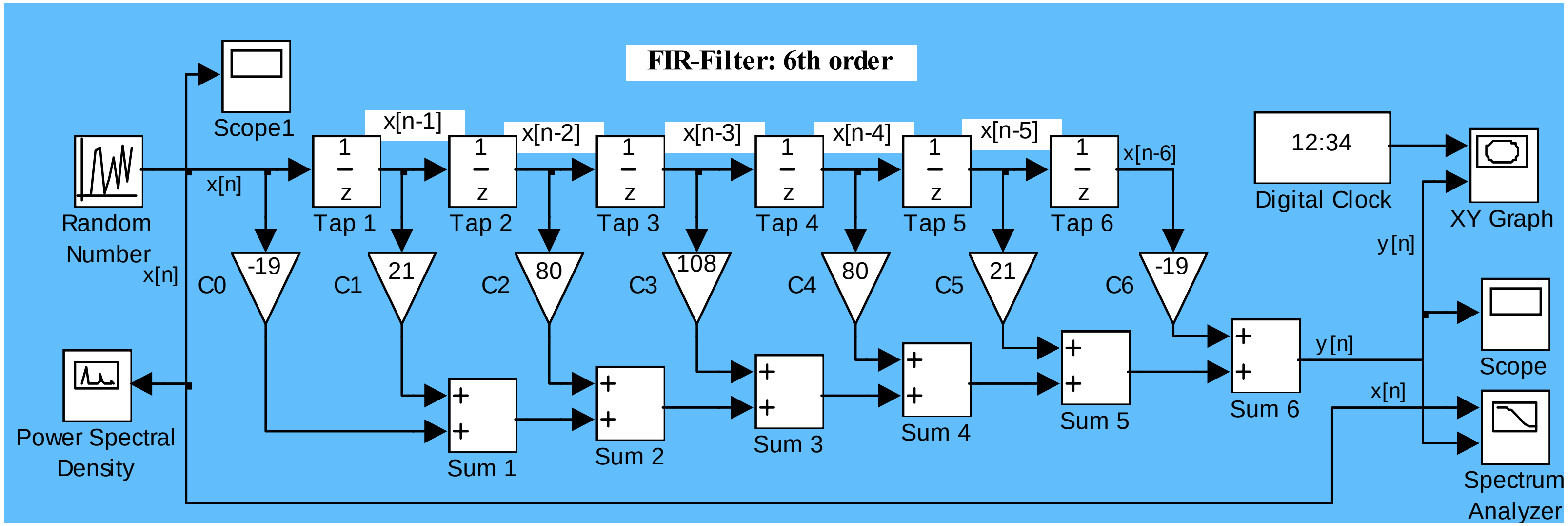


Fig. 2-9 : Simulink block diagram of a 6th order FIR lowpass.

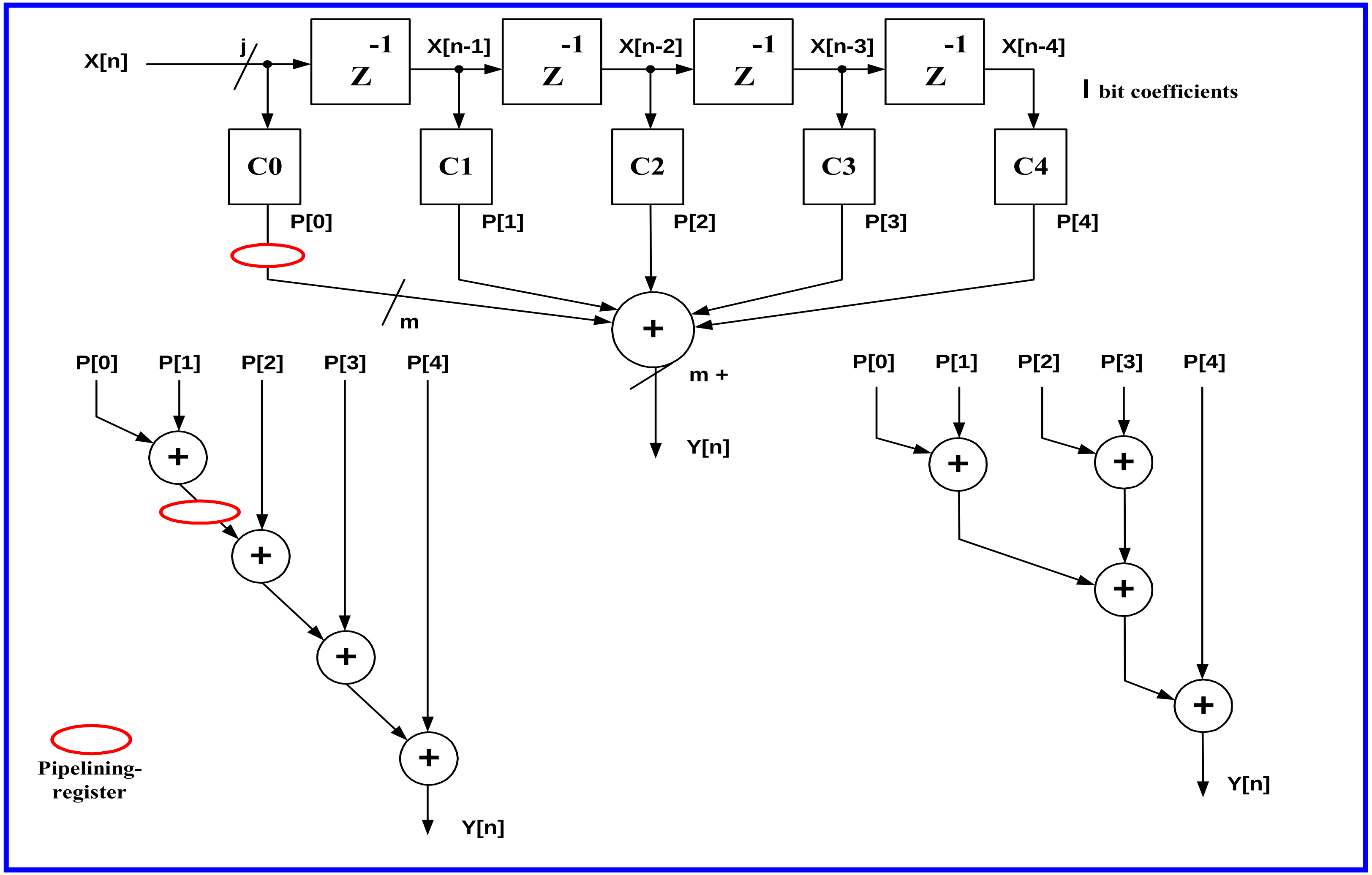


Fig. 2-10: Balanced adder tree with less signal delay stages.

- **Hardware implementation of direct form I under clock frequency constraints:**

➢ **The adder tree should be balanced by adding pairs of products. Further adder stages will combine pairs of added products. The number of adders will remain the same but a shorter signal delay path through the adders will result. The parallel structure of a balanced adder tree reduces the number of delay stages from N to ⌈N/2.**

➢ **The VHDL coding of balanced adder trees for higher orders N with `for`-loops or generated component instantiation will become more complicated in case of required parameterised solutions.**

- **Separation of adder stages by register will allow a higher clock frequency. The products can be stored and a register is assigned to every adder stage.**
- **In case of a balanced adder tree pipelining will need less registers and less clock cycles (smaller latency) are necessary for each new updated filter result.**
- **With $f_S << f_{clock}$ the products are constant during a sample interval so that no registers for delay balancing are needed.**

2.3.2 Direct Form I with Reduced Number of Multipliers

- Symmetry in FIR filters can be utilised for a reduced amount of implemented hardware. In case of positive coefficient symmetry the number of multipliers decreases from N+1 to N/2+1 because multiplicants are equal in pairs. The number of adders remains the same.
- In comparison to the basic direct form I the longest signal delay path is shorter because only N/2+1 adders have to be processed. Decoupling of combinational logic with registers is possible too.

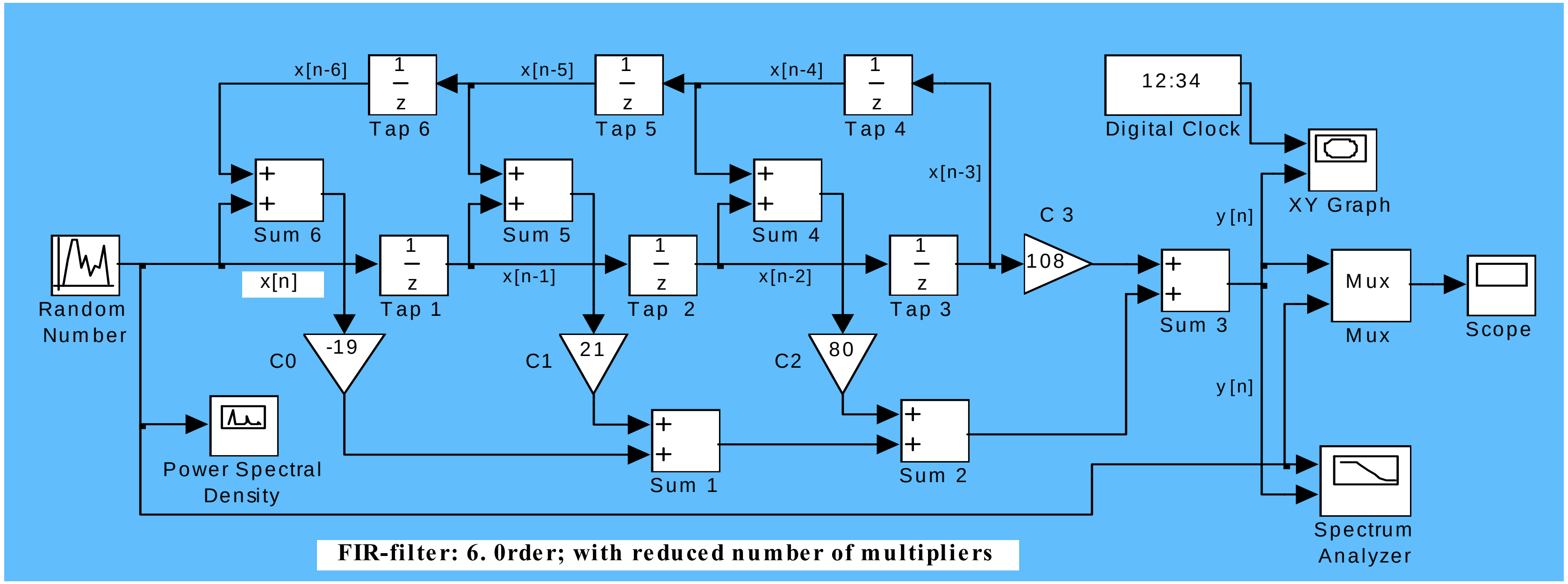


Fig. 2-11: Simulink block diagram of a direct form I lowpass with reduced number of multipliers.

2.3.3 Transposed Form

The transposed form is derived from the direct form by several manipulations of the filter structure:

- **Interchange input and output.**
- **Reversal of signal flow.**
- **Substitution of adders by a branch and vice versa.**

- **The main advantage of this structure is provided by the double use of delay stages because the registers directly decouple the adder stages. No additional pipelining stages for the adder tree are needed.**
- **The number of D-FFs encreases because now products with $m = j + l - 1$ bits are registerd.**
- **The longest signal delay path is made up by a multiplier (coefficient C0) and the last adder stage which contains the longest ripple carry chain with $(m + \log_2(N+1))$ bits.**

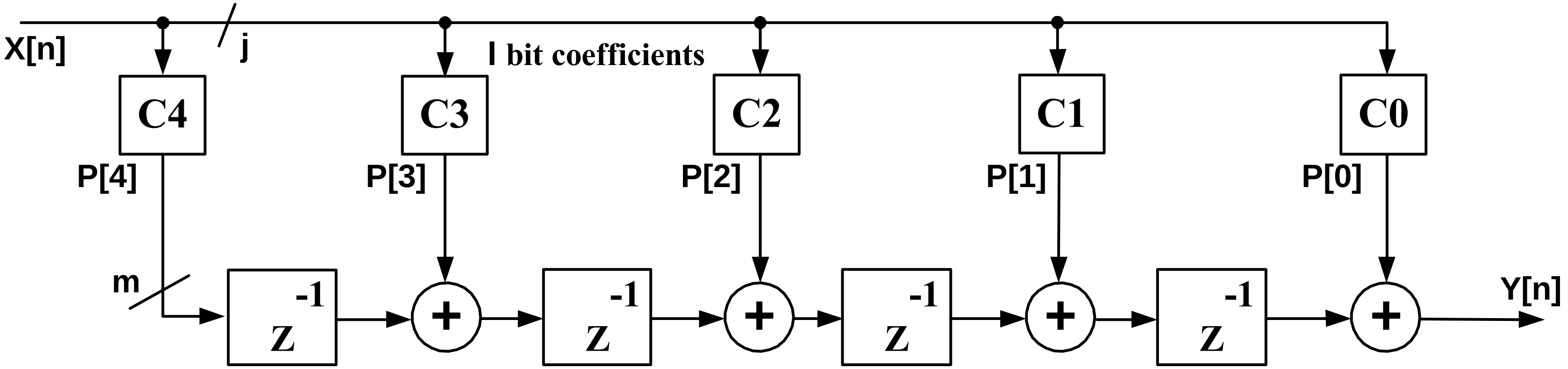


Fig. 2-12 : Transposed form with implicitly pipelined adder stages.

2.4 Fixed Point arithmetic implementation with VHDL and Filter Scaling

- System designs based on FPGA and ASIC hardware implementation are represented with VHDL bit level descriptions. With `unsigned` and `signed` signal types arithmetic operations are supported by the `IEEE.std_logic_arith.all` library. Therefore only integer- und fractional number representations are avaiable for two's complement arithmetic.

- Integer string with word length B+1: $x_{Bin} = b_B\, b_{B-1} \ldots b_1\, b_0\bullet$ und b ist ∈ (0,1). Negative numbers with sign bit $b_B = 1$.

$$\boxed{x_{Dec} = -b_B \cdot 2^B + \sum_{i=0}^{B-1} b_i \cdot 2^i}$$

- Fractional numbers $x_{Bin} = b_B\bullet\, b_{B-1} \ldots b_1\, b_0$ represent values within the range $-1 \le x < 1$.

$$\boxed{x_{Dec} = -b_B + \sum_{i=1}^{B} b_{B-i} \cdot 2^{-i}}$$

2.4.1 Fractional Binary Numbers; Q Format

- The fractional representation will be preferred for samples and coefficients because a multiplication of fractional multiplier and multiplicand will always provide limited results which are less/equal 1. The result grows downwards in the direction of the LSB.
- The Q notation is commonly adopted for specifying the position of the implied radix, so that a binary number in QB format is defined as having B bits to the right of the radix (binary point). So for example Q7 format has 7 bits to the right of the radix and 8 bits are required to hold a Q7 number. The implied radix is just shifted to the left by B bits:

QB: sign bit•B significant bits

- Example: Q7 signed binary fractional number

$$01000011_{bin} = (0*sign + 1*1/2 + 0*1/4 + 0*1/8 + 0*1/16 + 0*1/32 + 1*1/64 + 1*1/128)_{dec}$$
$$= (0.5 + 0.015625 + 0.\ 0078125)_{dec} = 0.5234375_{dec}$$

- **The negative number –0.5234375 with Q7 representation is derived by applying the one's complement and adding a one to the LSB position: 1.0111101$_{\text{bin}}$**

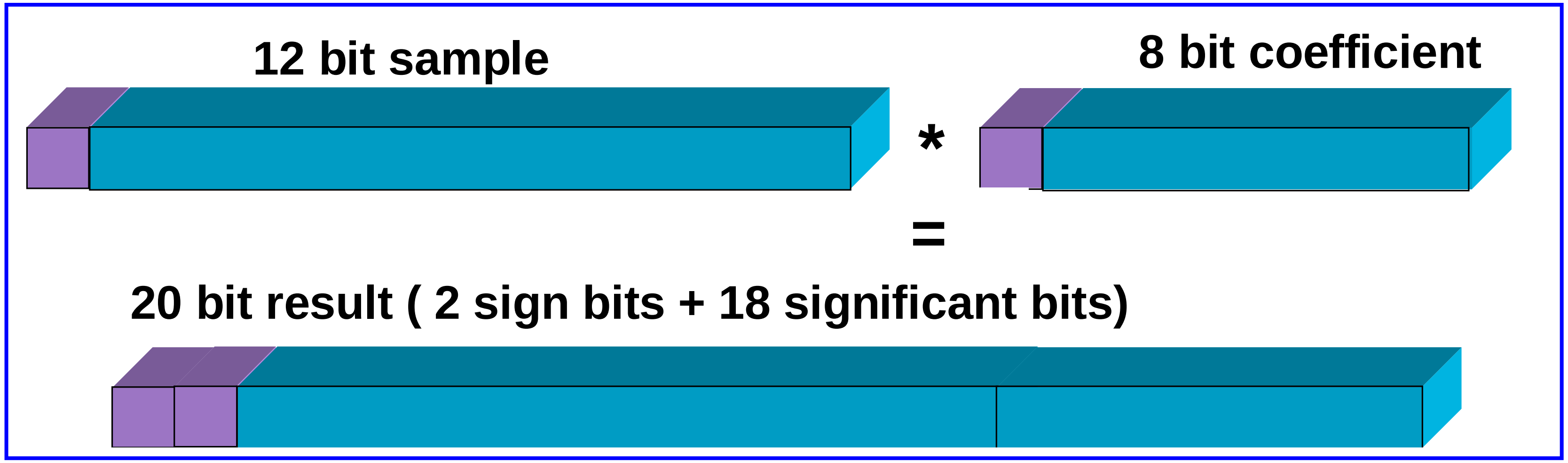
$$\boxed{-x_{Dec} = -\bar{b}_B + \sum_{i=1}^{B} \bar{b}_{B-i} \cdot 2^{-i} + 2^{-B}}$$

- **Range of values which can be presented by the QB format with word length B+1 are as follows:**

Word length	B+1	8	10
Quantisation step size (LSB)	D = 2^{-B}	2^{-7}= 0.0078125	2^{-9}= 0.001953125
Maximum positive value (1 – LSB)	1 – 2^{-B}	0.9921875	0.998046875
Minimum negativ value	-1	-1	-1
Dynamic range = Max/Min	$2^{B} – 1 \approx 2^{B}$	128 –1 = 127	512 –1 = 511
Accuracy	D/2 = $2^{-(B+1)}$	0.00390625	0.000976562

2.4.2 Multiplication with Q Format Multiplier and Multiplicand

- A multiplication with a QB_1 multiplicand and a QB_2 multiplier will provide a QB_3 result with $B_3 = B_2 + B_1$ bits right from the radix but in total with a string of $(B_3 + 2)$ bits.
- Each Q format multiplication result includes two sign bits where the left most is a “real” one and the sign bit next to the radix can interpreted as a so called guard bit.
- Filter structures are always build up with multiplications which are feed into a an adder tree. Therefore the guard bit can be used for capture of a temporary integer value.



```
                  1 0 0 0 0 0 0 1  (-0.9921875)  =  x(k)
           X      0 0 0 0 0 1 0 1   (0.0390625)  =  h(0)
    ------------------------------------------
              1 1 1 0 0 0 0 0 0 1
            + 0 0 0 0 0 0 0 0 0
            ---------------------
            1 1 1 1 0 0 0 0 0 0 1
          + 1 1 0 0 0 0 0 0 1
          -----------------------
          1 1 0 1 1 0 0 0 0 1 0 1
        + 0 0 0 0 0 0 0 0 0
        -------------------------
        1 1 1 0 1 1 0 0 0 0 1 0 1
      + 0 0 0 0 0 0 0 0 0
      ---------------------------
      1 1 1 1 0 1 1 0 0 0 0 1 0 1
    + 0 0 0 0 0 0 0 0 0
    -----------------------------
    1 1 1 1 1 0 1 1 0 0 0 0 1 0 1
  + 0 0 0 0 0 0 0 0 0
  -------------------------------
  1 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1
+ 0 0 0 0 0 0 0 0 0
  -------------------------------
  1 1 1 1 1 1 0 1 1 0 0 0 0 1 0 1  (FD85 = -0.38757324 )
```

1,0 = SIGN EXTENSIONS

2.4.3 VHDL Model of a FIR Filter with Direct Form I

- **Constant coefficients and stage registers are declared as Array of signed vectors which are supported by two's complement arithmetic.**
- **Samples are stored with directly connected register stages which are inferred by a clocked process. Shifting within the register chain is modelled by a `for-loop`.**
- **With a `READY` pulse which is sent by the receiver register of the codec-FPGA interface the capture of a new sample X and shift enable for the register chain is performed. The first register stage is an additional tap which operates as an take over register.**

- **With filter order N = 6 7 products have to be generated and added. This is performed by a `for-loop` which provides signal processing with VHDL variable support for products and partial sums of the adder tree.**
- **The adder variable `Y_VAR` is initialised with zero in order to avoid unnecessary D-FFs so that only the final sum value is stored with a hand over register Y.**
- **Arithmetic operations are processed within one clock cycle which is enabled after `READY` pulse has passed. Updating of filter output Y is performed within two clock cycles.**

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

entity FIR_VAR_2 is
          generic
          (    DAC_WIDTH: positive := 20;
               ADC_WIDTH: positive := 20;
               TAPS: positive := 6);
          port
          (    CLK: in bit;        -- clock input 12MHz
               RESET: in bit;      -- synchronous active-high reset
               READY: in bit;      -- Flag --> enable of filter operation
               X: in bit_vector(DAC_WIDTH-1 downto 0); -- input to the filter register stages
               Y: out bit_vector(DAC_WIDTH-1 downto 0)-- output
          );
end FIR_VAR_2;

architecture FIR_ARCH of FIR_VAR_2 is

--Filter with fractional sample and coefficient representation
type COEFF_TYPE is array(0 to TAPS) of signed(7 downto 0); -- coefficient array
type STAGE_TYPE is array(0 to TAPS) of signed(11 downto 0); -- register array
signal COEFF: COEFF_TYPE; -- coefficient vectors
signal STAGE: STAGE_TYPE; -- input signal register stages
signal FIR_EN: bit ;-- enable  filter operation only for one FIR_CLOCK cycle
----------------------Low pass filter 6kHz --> 12kHz -12 dB-----------------
begin
     COEFF(0)  <= "11110111";     -- - 0.0698529 filter amplification factor V = 1
     COEFF(1)  <= "00001001";     --   0.0735294
     COEFF(2)  <= "00100110";      -- 0.294117647
     COEFF(3)  <= "00110011";      -- 0.3970588
     …
SYNC_RÊADY: process(CLK)
begin
     if   CLK='1' and CLK'event  then
               if RESET='1' then
                    FIR_EN <= '0';
               else FIR_EN <= READY ; -- READY pulse delayed by one CLK cycle
               end if;
     end if;
end process SYNC_READY;
-- higher 12 bit of X into the registers
STAGES: process(CLK)
begin
     if   CLK='1' and CLK'event  then   --
               if RESET='1' then
                    for I in 0 to TAPS loop
                         STAGE(I) <= (others=>'0');
                    end loop;
               elsif READY = '1' then -- enable of loading and shifting
                    STAGE(0)<= signed(to_stdlogicvector(X(19 downto 8))); -- loading
                    for I in TAPS downto 1 loop
                    STAGE(I) <= STAGE(I-1);-- shifting
                    end loop;
               end if;
     end if;
end process STAGES;
```

```
--
MAC: process(CLK)
type PRODUCTS_TYPE is array(0 to TAPS) of signed(19 downto 0);
variable PRODUCTS: PRODUCTS_TYPE;       -- 2 sign bits and 18 significant bits
variable Y_VAR: signed(20 downto 0);    -- integer bit enhancement to avoid temporary overflow
begin
          if CLK='1' and CLK'event then --
               if RESET='1' then
                    Y <= (others=>'0');
               elsif FIR_EN = '1' then
               Y_VAR := (others=>'0');
                         for I in 0 to TAPS loop
                         -- Multiplication and accumulation
                              PRODUCTS(I) := COEFF(I) * STAGE(I);
                              Y_VAR := Y_VAR + (PRODUCTS(I)); -- implicite sign extension
                         end loop;
                    Y <= to_bitvector(std_logic_vector(Y_VAR(18 downto 0))) & '0'; -- shift left
-- Accu result will be registered
-- inner conversion used from IEEE.std_logic_arith.all; outer used form IEEE.std_logic_1164.all;
               end if;
          end if;
end process MAC;
------------------------------------------------------------------------
end FIR_ARCH;
```

2.4.4 Serial Structure with MAC Resource Sharing

- **If the data input rate is slower than the performance of the FPGA, the FIR filter convolution (comp. Fig. 2-13) can be implemented more efficiently than with parallel structures. Assuming that the clock frequency of the FPGA is M times higher than the data input rate (M > N +1) [10].**
- **To implement a serial N-tap filter uses only one multiplier, a 2-input adder and storage for the partial results and the filter input samples.**
- **The input sample storage holds the last N+1 input samples. For every new sample entering the filter N+1 multiply operations will be performed, each multiplying the filter coefficient by the respective input sample (comp. Fig. 2-14).**
- **The result of each multiplication is accumulated to the partial result storage to produce a new partial result. After N+1 such multiply and add operations the partial result is driven out of the filter.**
- **The accumulator storage content is then cleared to begin processing a new data sample.**
- **The input data storage can be implemented as a shift register which holds N+1 samples. With each new sample shifted into the register the oldest is shifted out. Then the samples will circulate around the shift register supplying the MAC unit with multipliers. In parallel appropriate coefficients are supplied to the MAC as multiplicands.**

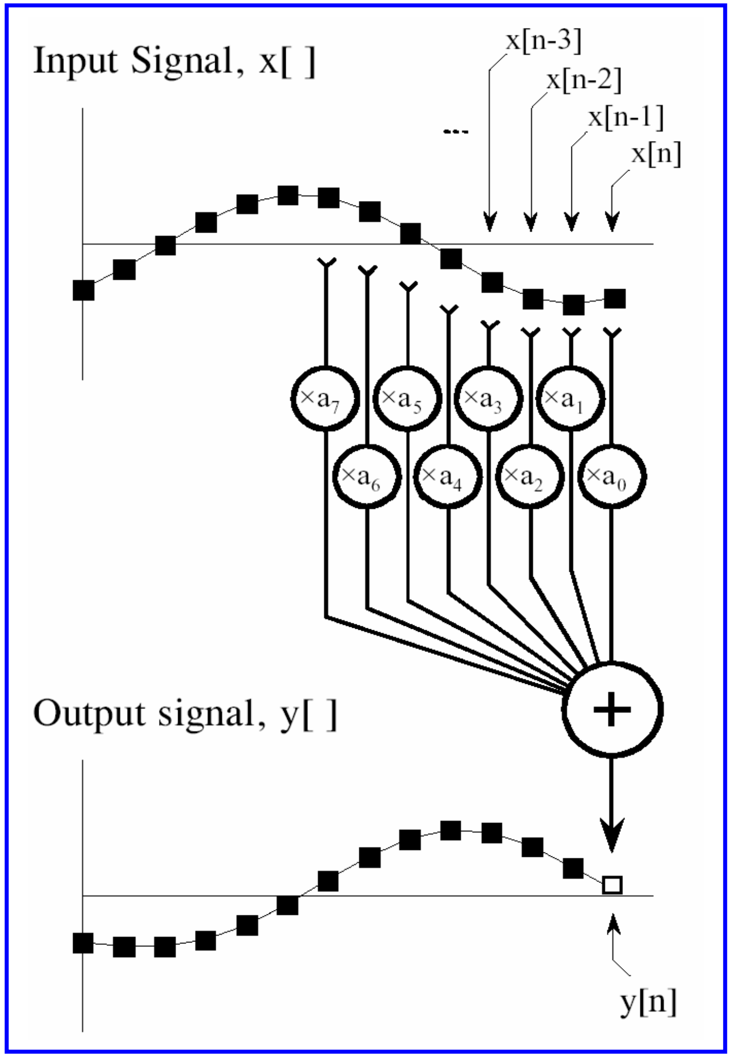


Fig. 2-13: FIR filter convolution [2, chapt. 28].

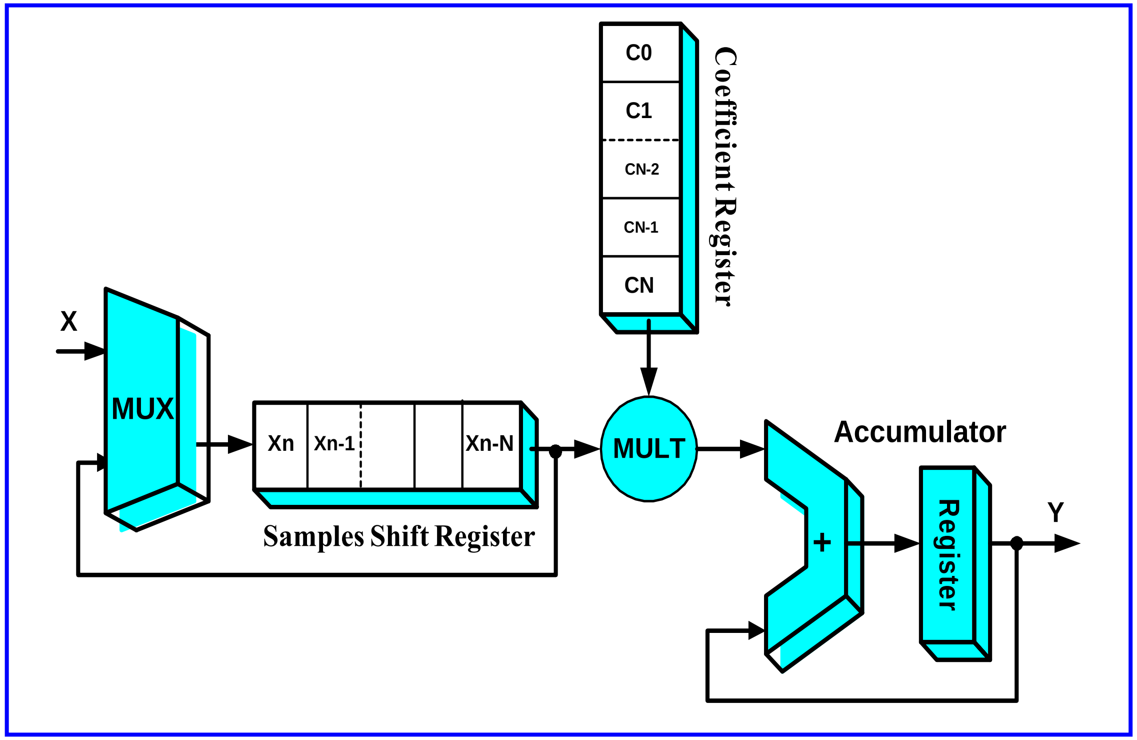


Fig. 2-14: Serial FIR filter structure with samples shift register and multiply and accumulate (MAC) unit.

- **An alternative approach which is suggested for the first laboratory task will avoid registers which are realised with D-FFs. Especially Spartan and Virtex FPGAs offer separate Block RAM modules (comp. Chapt. 5) which will be used as efficient storage elements for samples and coefficients.**
- **This will give the combinational logic elements (CLBs = 2 slices) and D-FFs free for the convolution algorithm and higher filter orders N > 64 can be realised.**

- **The detailed block diagram in Fig. 2-15 describes a basic FIR filter system with 4 components:**
  - **MAC unit with 8 bit coefficient and 12 bit samples**
  - **Samples Block RAM library module RAM B4_S16 with address counter.**
  - **Coefficients Block RAM library module RAM B4_S8 with address counter.**
  - **Finite state machine controller.**

- **The strategy for addressing the RAM's contents in order to create appropriate pairs for the MAC unit input determines the state's sequence of the FSM. It is the aim to have simple parallel counting sequences which will support an easy to understand state machine with as few state transitions as possible.**

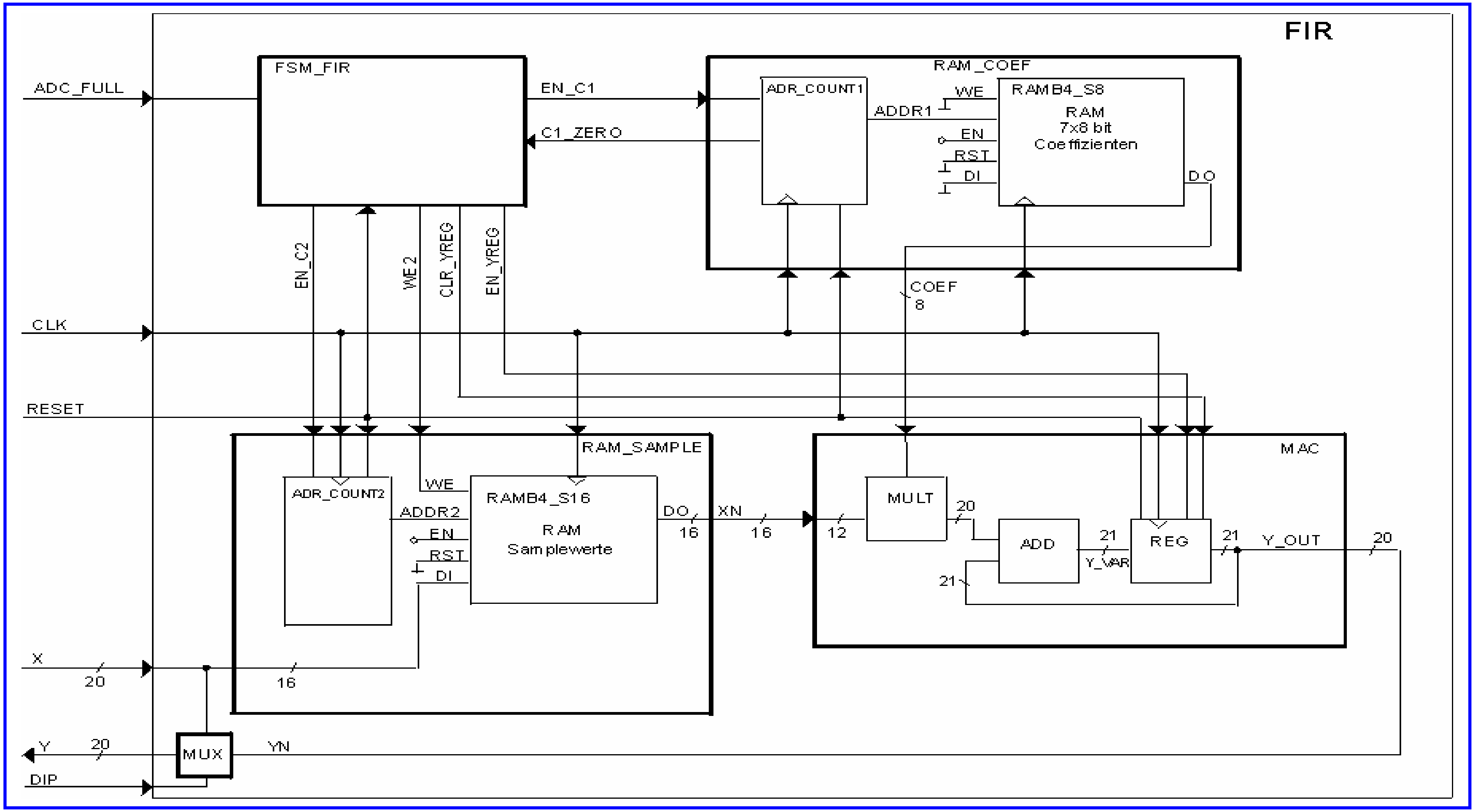
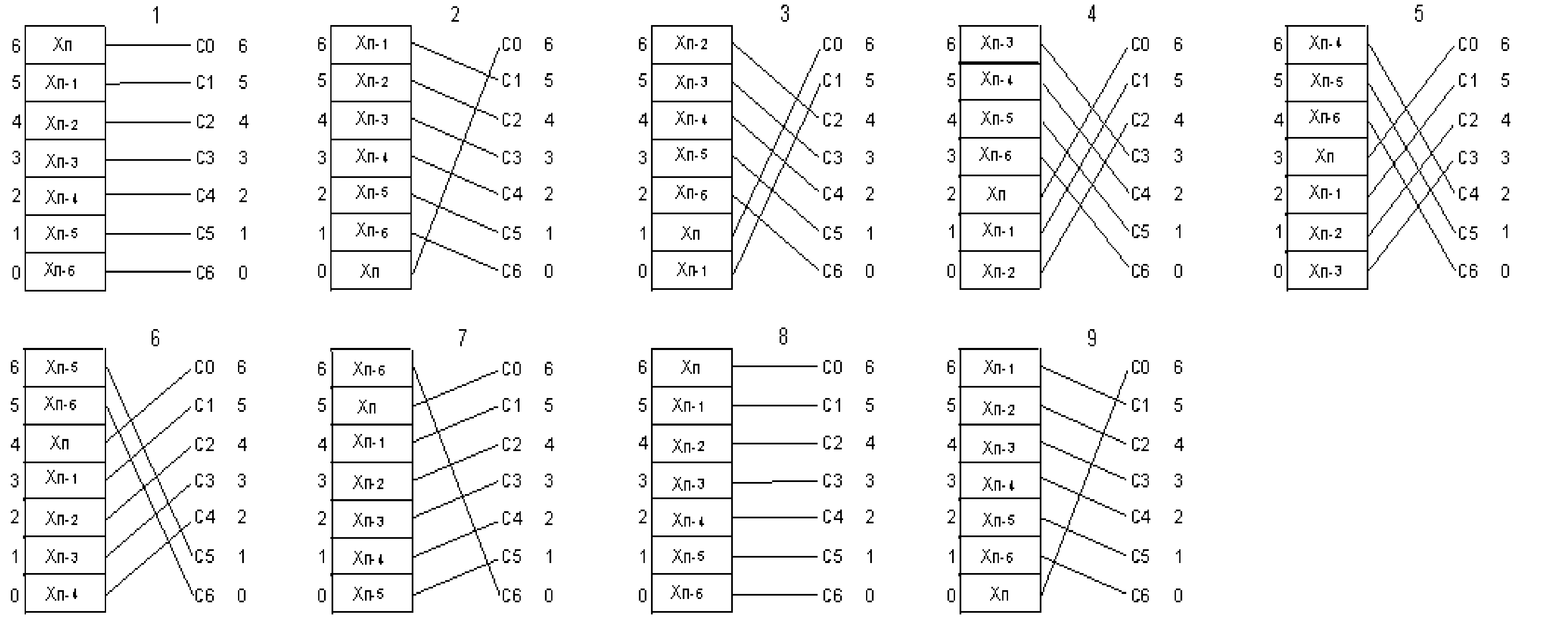


Fig. 2-15: Block diagram of the serial FIR filter structure with circular buffer.

- Characteristics of the addressing strategy suggested in Fig. 2-16:

  - Nine cycles of FIR filter output calculation are depicted with filled sample RAM cells and assigned coefficients (connection lines).
  - The numbers aside the samples x_{n-i} and the coefficients c_i assign the address of the corresponding RAM cell.
  - The sample RAM is filled beginning from the bottom at address zero.

    The coefficient RAM is filled from the top (highest address N) with c_0.
  - Both address counters are sequencing form zero to the highest value and start from the beginning.
  - Each filter cycle starts with over writing the oldest sample with the newest sample update: circular buffer principle.
  - In cycle 2 a new x_n is stored at address zero and coefficient address counter is zero too. The sample address counter will be incremented while the other is stopped. The first multiplication is performed with $c_6 * x_{n-6}$ and the last will be $c_0 * x_n$ while the address counters do a final increment.
  - Cycle 3 is initialised by an enable pulse from the codec (ADC_FULL, comp. Fig. 2-15). The new sample x_n is written to the RAM at address 1 and the counters are controlled in the same kind as at the beginning of cycle 2.



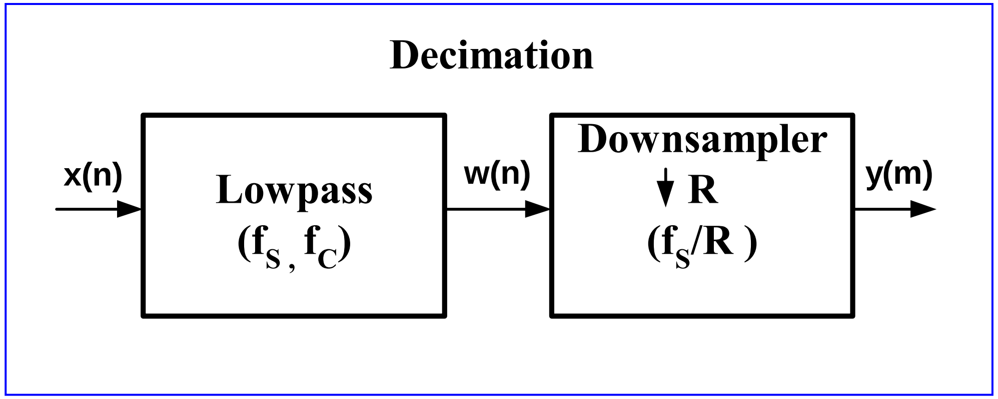
	schreiben								schreiben								schreiben								schreiben							
	2								3								4								5							
SSR-Adresse	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3
Coeff-Adresse	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6

	schreiben								schreiben								schreiben								schreiben							
	6								7								8								9							
SSR-Adresse	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0	1	2	3	4	5	6	0
Coeff-Adresse	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6	0	0	1	2	3	4	5	6

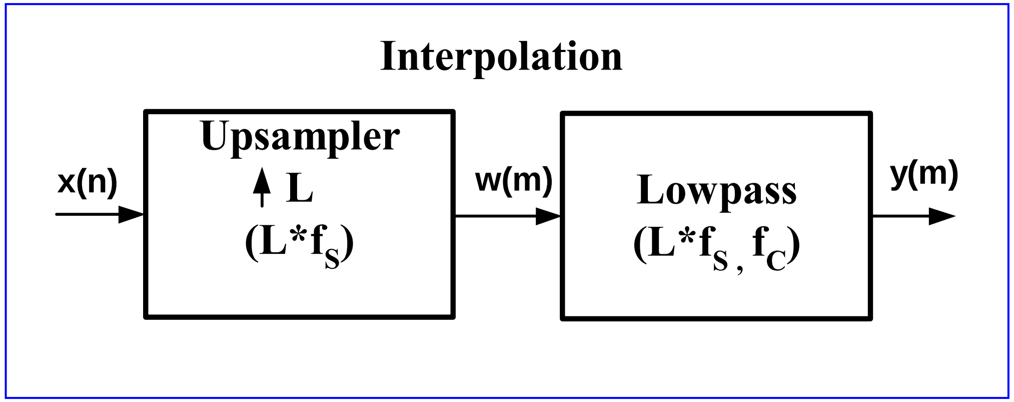
Fig. 2-16: Strategy for building pairs of samples and coefficients. Table with address counter correspondence for samples and coefficients. Example with filter order N = 6.

2.5 Concepts of Multirate Signal Pocessing with Parallel Filter Structures

- **Oversampling implemented with AD-conversion**
  - **A simple analog lowpass filter can be chosen to avoid aliasing and fewer bits for digital representation are needed.**
  - **The quantisation noise of the ADC process is shifted to a higher frequency band and can be attenuated with a digital lowpass filter.**
  - **The output of the digital lowpass can be sampled with a reduced sample frequency according to the sample theorem.**
  - **This decimator (lowpass and downsampler) adjusts the sampling frequency according to the signal of interest.**



- Interpolation of a digital output preceding to DA-conversion
  - ➢ Oversampling with an increased sample rate operates with inserted zeros to the sequence of samples.
  - ➢ A lowpass filter has to eliminate the image frequency components before DAC.
  - ➢ A simple analog lowpass filter for removing the harmonics from the step-like out put function can be used.
  - ➢ The quantisation noise of the DAC process is shifted to a higher frequency band.



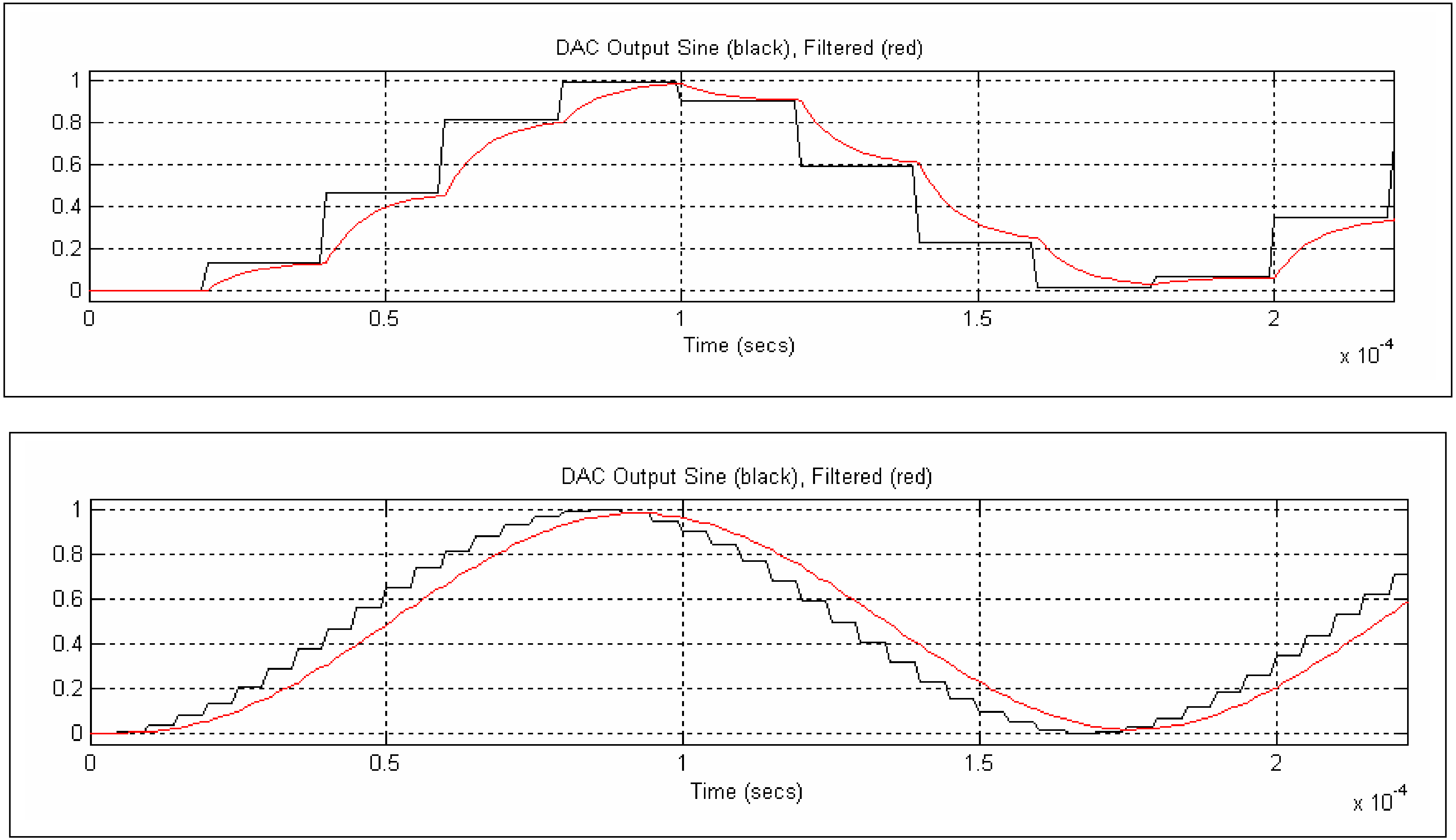


Fig. 2-17: Output of a zero-order hold with T_S = 20µs and T_S = 5µs. Lowpass filter 1rst order f_C=24kHz.

- **FIR filter structures offer implementations with efficient decimator and interpolator designs.**

  - **Problem: Narrow band FIR lowpass filters have to be implemented with a high filter order which means a large number of coefficients with an appropriate precision (word length).**

  - **Solution: A cascade of lowpass filter stages each with a reduced sample frequency.**

    The transition range $\Delta f = 3.3 f_S/(N+1)$ of the filter stages will be reduced with decreasing f_S.

    A smaller order N of each filter stage can be chosen in order to achieve a narrow Δf.

    That results in a reduced number of coefficients which need less word length.

    The overall number of coefficients within the filter cascade decreases in comparison to a direct solution.

2.5.1 Decimation

- **Downsampling is a part of each digital input stage where the ADC process is implemented with oversampling.**
- **Decimation is performed by an ordered discarding of waveform samples with the effect of reducing the effective sampling rate.**
- **For the subsequent digital signal processing the sampling rate should be minimized in order to**
  - ➢ **reduce the requirements for maximum allowed signal delay in combinational logic path**
  - ➢ **support precision with wider vector width for filter coefficients and**
  - ➢ **allow more guard bits in intermediate addition results.**
- **It is essential that the Nyquist constrained has still to be satisfied.**
- **Therefore additional filtering has to be performed digitally within the DSP chain prior to decimation.**

- **The input sampling results are depicted in Fig. 2-18 (comp. Code 2-6).**
  - ➢ **The output of the multiplication process (sampling with dirac pulses δ(n)) has a spectrum consisting of components spaced at the sum and differences between input f_1, f_2 and sampler spectral lines at f_S.**

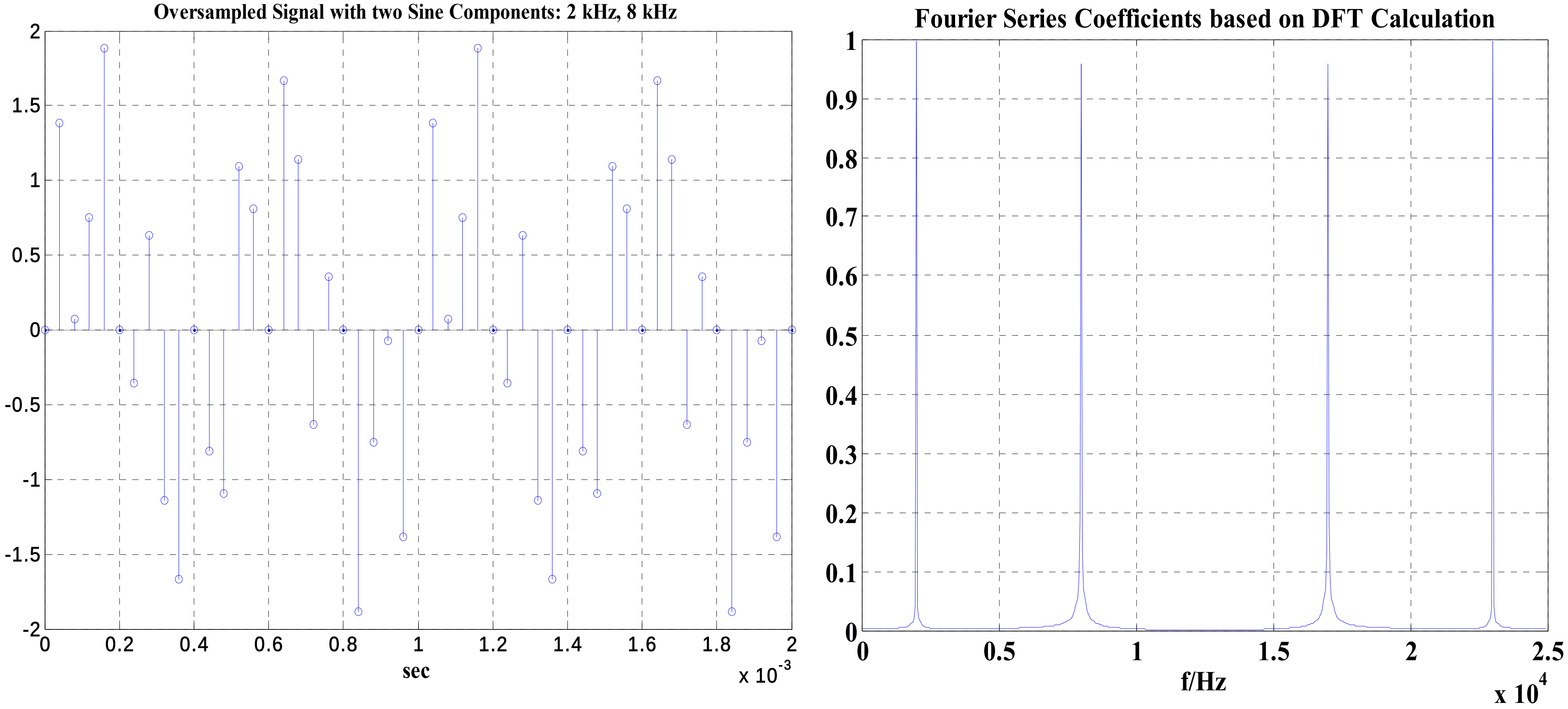


Fig. 2-18: Oversampled input signal s(n) (f_S = 25 kHz) a) and fourier series coefficients $A_k = 2|X_k|/m$ based on DFT calculation X_k for m = 512 points (input samples) b).

- **With each decimation it is necessary to ensure that the signal to be decimated occupies a bandwidth no greater than $0.5f_S/R = f_N/R = f_{Nnew}$ (R sampling reduction factor).**

- **For the chosen example (comp. Fig. 2-18, Fig. 2-19) with the factor R = 2 this requires that the signal prior to decimation becomes the subject of lowpass filtering. All frequency components greater than $f_{Nnew} = f_S/4$ ideally should be attenuated.**

- **The selected filter's frequency response with $f_C = 0.3f_N < f_{Nnew}$ satisfies this condition. Only the 2kHz sine fits to the passband and the 8 kHz sine is totally extinguished. With the parameters of this example the filter's magnitude response provides a zero at 8 kHz.**

- **For direct calculation of the input signal's spectrum a DFT would be the choice because it provides an approximation of the Fourier series coefficients. Matlab offers only the FFT as the more efficient method which needs much less operations for m input samples**

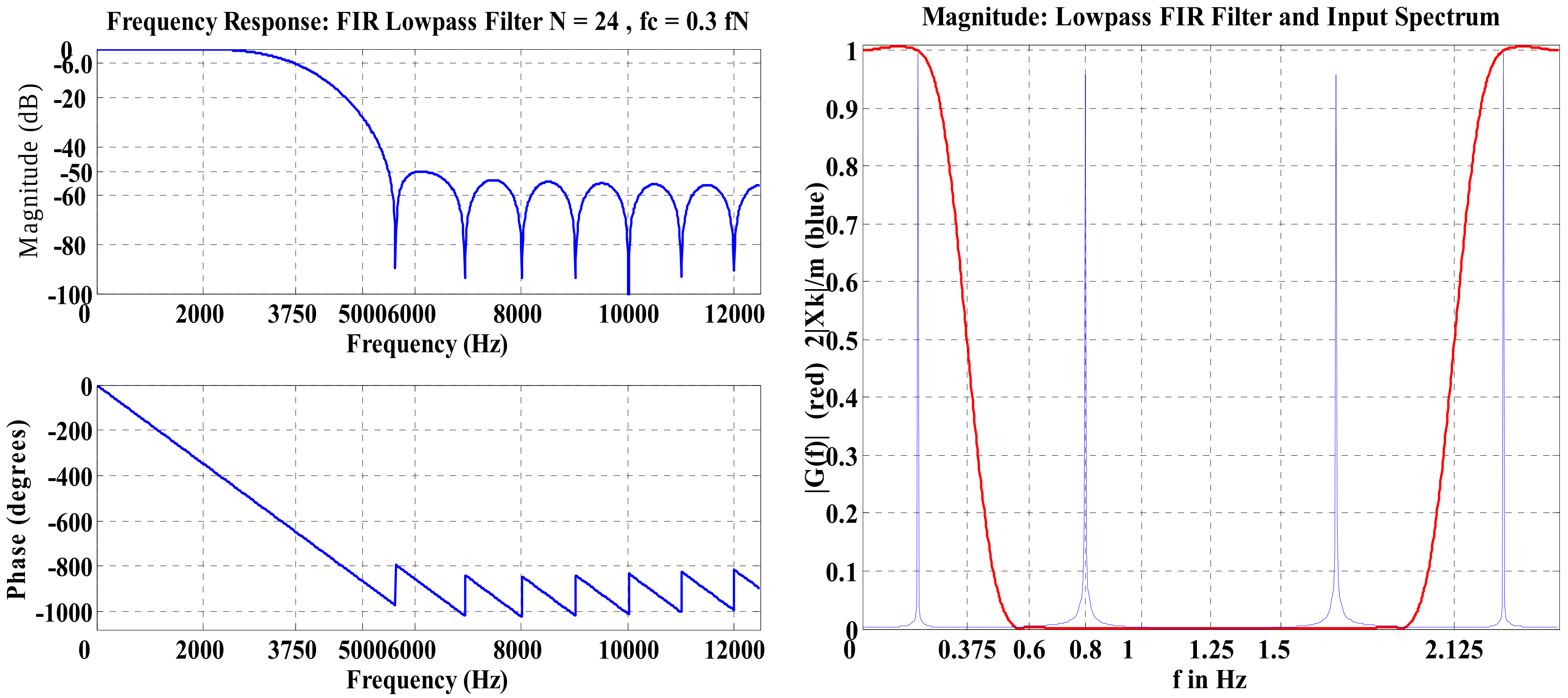
- **In Code 2-6 the sampling parameters are chosen as follows:**

  m is an integer number of periods for both frequency components.

  Number m = 2^9 = 512 as a power of 2.

```
%Decimation; Frequency components are estimated with DFT
Ts =0.00004; fs = 1/Ts;     % sample periode fs =  25kHz
m = 512; % number of signal samples
% With periodic signals an integer number of periods should be covered: Approximation aspects
% inorder to apply the FFT m should be power of 2
t = 0:Ts:(m-1)*Ts;    % vector like linspace
x = sin(2*pi*2000*t)+sin(2*pi*8000*t);     % y = x + 0*randn(size(t));
figure(1)
tshort = 0:Ts:(m/10-1)*Ts;
stem(tshort,x(1:m/10));
axis([0 m*Ts/10 -2 2]);  grid;    title('Oversampled signal with two …')
%Converting to the frequency domain, the discrete Fourier transform
%of signal x is found by taking the m-point fast Fourier transform (FFT):
X = fft(x,m); % The DFT coefficient Xk is related to the fourier series coefficient Ak=2*|Xk|/m
Mag = 2*sqrt(real(X).*real(X) + imag(X).*imag(X))/m; f = (1/Ts)*(0:(length(X)-1))/length(X);
figure(2);
plot(f,Mag(1:m)); grid; axis([0 fs 0 1]);% original frequency range
```

Code 2-6: Oversampled signal s(n) = x and fourier series coefficient calculation.



Frequency Response: FIR Lowpass Filter N = 24 , fc = 0.3 fN

Magnitude: Lowpass FIR Filter and Input Spectrum

Fig. 2-19: Pre decimation lowpass filtering with $f_C = 3.75$ kHz. Decimation by R = 2, $f_{Snew} = f_S/R = 12.5$ kHz. (comp. Code 2-7)

```
%Lowpass filter design with windowing method, default Hamming window
N = 24; % filter order
fc = 0.30;                    % f = 1 Nyquist frequency = fs/2
h = fir1(N,fc);   % alternative boxcar(N+1) as 3rd parameter
figure(3);  freqz(0.99*h,1,1024,fs);     % direct plot of magnitude and phase
                                          % abs(G) = abs(fft(b,n)./fft(a,n))
figure(4);  [G,f] =freqz(h,1,1024,'whole',fs);      % w in Hz
plot(f,abs(G)); hold on; plot(f,Mag(1:m));
grid; axis([0 fs 0 1]);% original frequency range
hold off;
xlabel('f in Hz'), ylabel('|G(f)|  and  2|Xk|/m ')
```

Code 2-7: FIR filter coefficients h_k. Plot of magnitude response and spectral components of input signal.

```
%Bandlimiting of the input samples before decimation
xf = filter(h,1,x);
figure(5)
stem(tshort,xf(1:m/10));
axis([0 m*Ts/10 -1 1]); grid;
title('Filtered Signal')
XF = fft(xf,m);
figure(6)
plot(f,2*abs(XF(1:m))/m);grid ; title('Spectral Components');
```

Code 2-8: Lowpass filtering with f_C = 3.75 kHz and calculation of reduced spectral components for f_S = 25 kHz.

- **The *filter* function filters a data sequence using a digital filter which works with a direct form II transposed implementation of the standard difference equation.**

$$xf(n) = h(0)*x(n) + h(1)*x(n-1) + \ldots + h(N)*x(n-N)$$

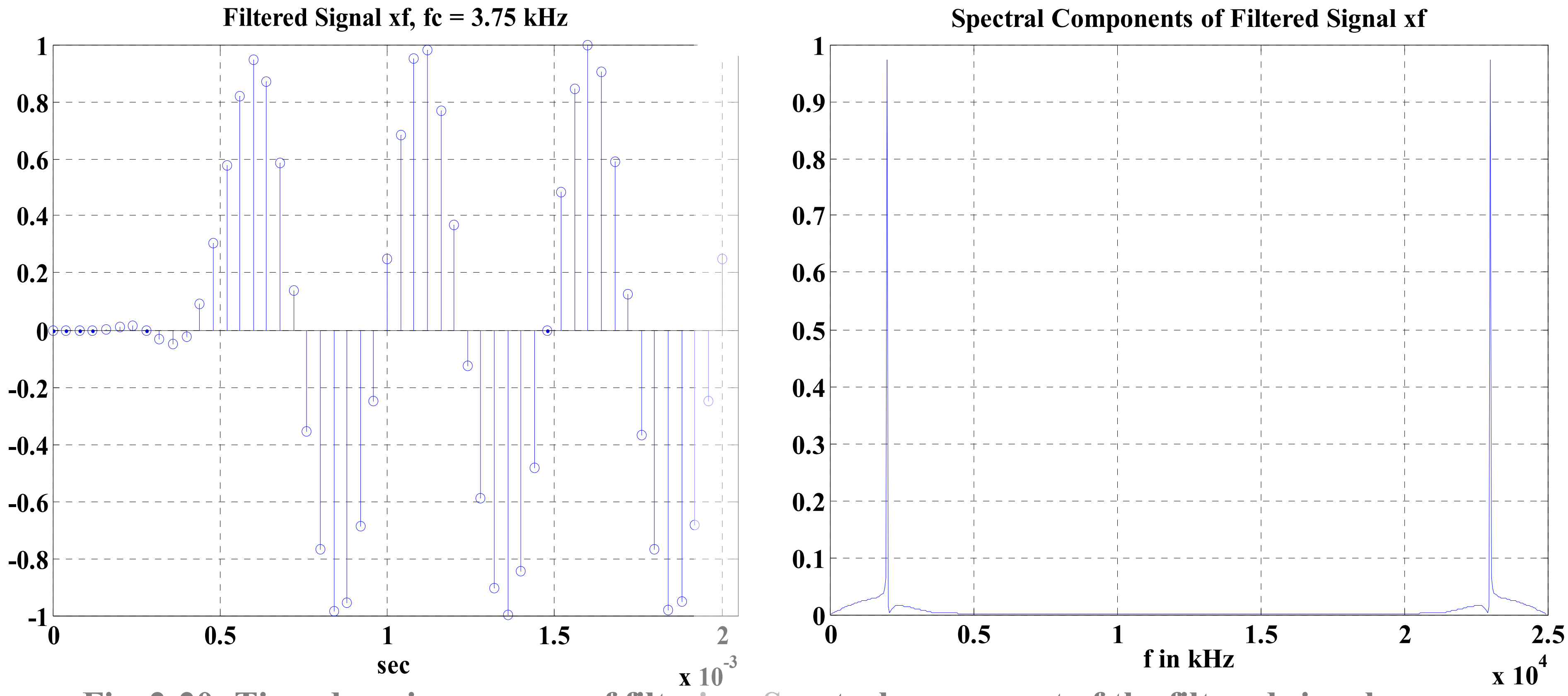


Fig. 2-20: Time domain response of filtering. Spectral component of the filtered signal.

```
% decimation to fsnew = fs/2: discarding every second sample
for i=1:m/2
  xd(i)=xf(2*i-1);% only odd index of xf is used -> reduced number of samples m/2
end;
XD = fft(xd,m/2);      %The spectral components 2|Xk|/(m/2)
MXD = 2*sqrt(real(XD).*real(XD) + imag(XD).*imag(XD))/length(XD);
fd = 0.5*(1/Ts)*(0:(length(XD)-1))/length(XD); % new frequency range fsnew
figure(7);
plot(fd,MXD(1:length(XD)));grid;
axis([0 fs/2 0 1]);% decimated frequency range
figure(8)
tshort = 0:Ts*2:(m/10-1)*Ts;  %same time range but increased sample period 2*Ts
stem(tshort,xd(1:round(m/20)));
axis([0 m*Ts/10 -1 1]);grid;   title('Decimated Signal')
```

Code 2-9: Decimation after bandlimiting by lowpass filtering.

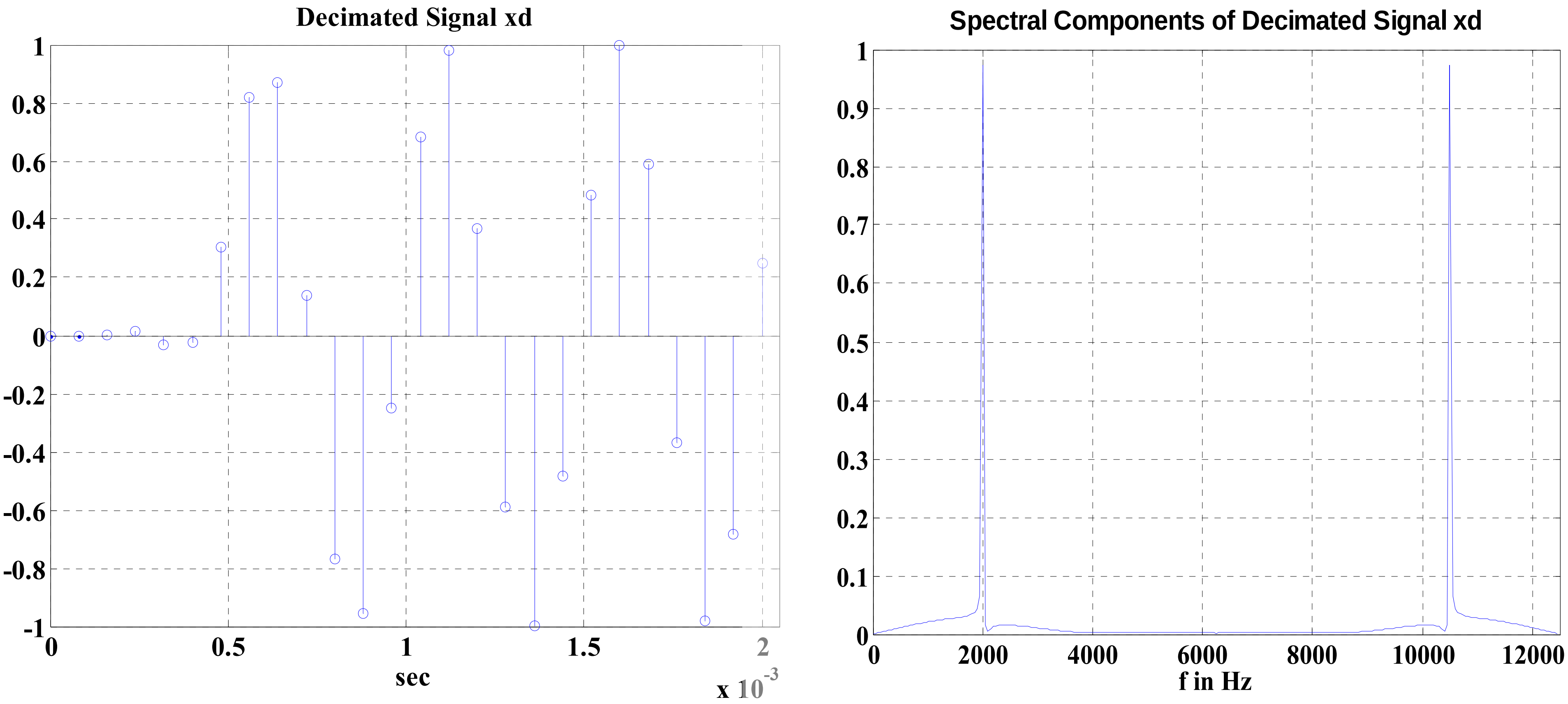


Fig. 2-21: Decimation with R = 2, f_{Snew} = 12.5 kHz. Time domain and spectral components.

2.5.2 Interpolation

- It is often desirable to increase the sampling rate before DA conversion to permit reduced requirements for the analog reconstruction filter.

- The process is achieved by inserting new samples between existing samples. There are two choices of sample values to intersperse:
  - Sample values equal to the most recent existing sample.
  - Sample values of zero.

- It is most common to use interpolation with zero values because it simplifies the filter implementation. Equal values result in greater signal energy in the interpolated signal.

- A lowpass filtering is necessary to complete the interpolation process (comp. Fig. 2-22):
  - The desired sample values in between the original waveform samples are found by ‘averaging’ using a filter which closes the gaps filled with zeros.
  - The new zero-valued gaps introduce higher frequency components which do not match to the original spectral composition.
  - By increasing the sample rate the former ‘alias’ spectral components appear within the frequency range $f_{Snew}/2 = f_{Nnew}$ and therefore have to be extinguished.

```
% Interpolation process with explecitely upsampling and lowpass filtering
Ts =0.00004;  fs = 1/Ts;     L = 2; % sample periode fs = 1/TS = 25kHz, upsample rate fup = L*fs
m = 512                    % number of signal samples
for i = 1:L*m              % increased number of samples L*m, new vector xup initialised with zeros
  xup = 0.0;      end;
for i=1:m
  xup(2*i-1)=x(i);  end;     % original samples at odd indices, inserted zeros at even indices
XUP = fft(xup,L*m);        %Magnitude approximation of the Fourier coefficients
MXUP = 2*sqrt(real(XUP).*real(XUP) + imag(XUP).*imag(XUP))/length(XUP);
fup = L*(1/Ts)*(0:(length(XUP)-1))/length(XUP);     % enhanced frequency range
figure(3); tshort = 0:Ts/L:(floor(m/10)-1)*Ts;  %same time range, decreased sample period Ts/L
stem(tshort,xup(1:floor(L*m/10)-1));       % number of elements can be checked with workspace
axis([0 (m-1)*Ts/10 -2 2]); grid;   title('Interpolated Sine Signal, L*fs = 50kHz')
xlabel('t in s'), ylabel('  ')
figure(4);  plot(fup,MXUP(1:length(XUP))); grid;
axis([0 fs*L 0 0.5]);      % increased frequency range fs*L, reduced amplitude by 1/L !
xlabel('f in Hz'), ylabel('2|Xk|/(L*m)'); title('Fourie Series Coefficiets…')
```

Code 2-10: Upsampling with inserted zeros to the original signal. Resulting spectral composition in the frequency range up to Lf_S.

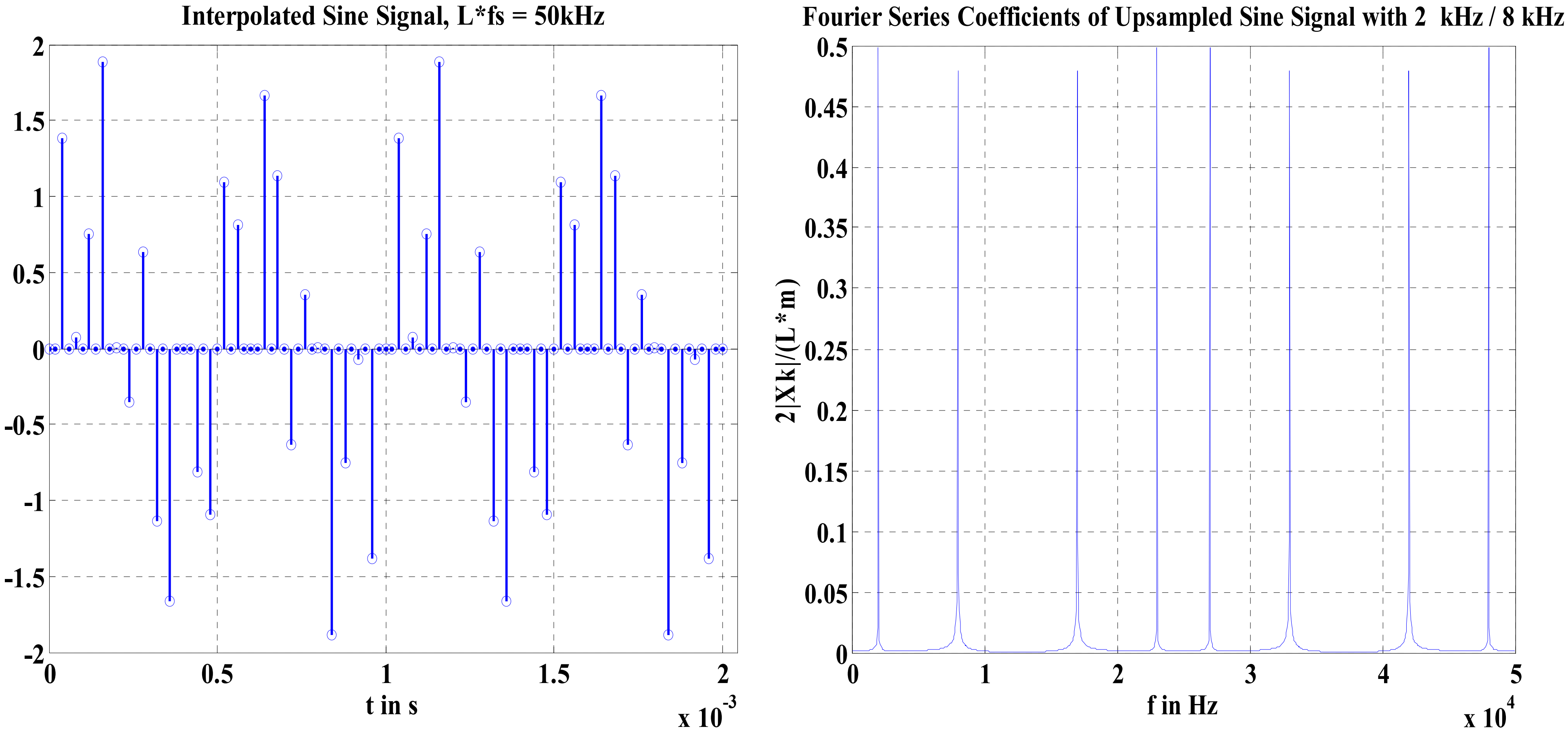


Fig. 2-22: Interpolation in the time domain with inserted zeros (f_{Snew} = 50 kHz, L =2). Spectral composition with additional 'alias' components and reduced magnitude (1/L).

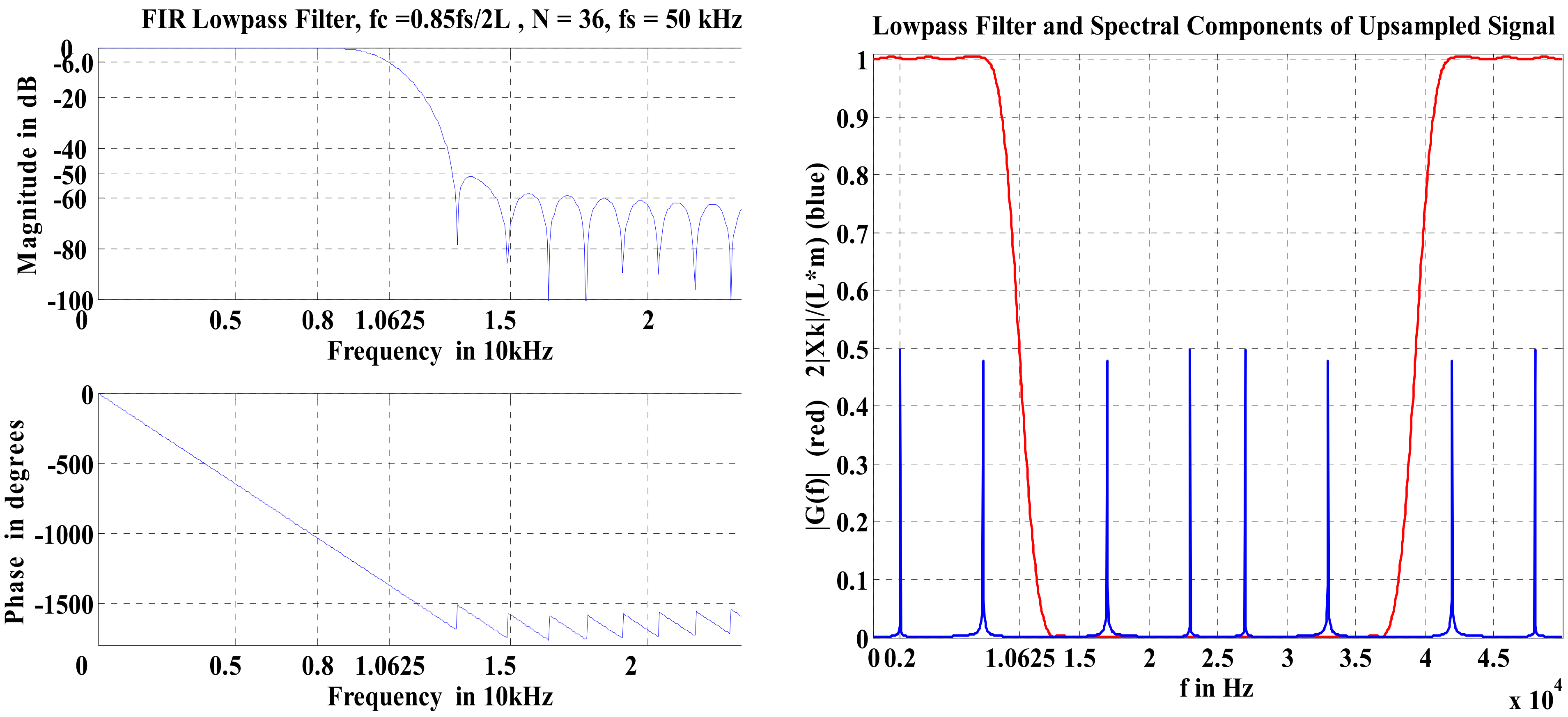


Fig. 2-23: Interpolation lowpass filtering with $f_C = 10.625$ kHz. Upsampling with L = 2, $f_{Snew} = f_S L = 50$ kHz. (comp. Code 2-11)

```
%Lowpass filter design with windowing method
N = 36;                  % filter order N
fc = 0.85/L;         % f = 1 upsampled Nyquist frequency = L*fs/2
h = fir1(N,fc);     % -6dB at  fc*L*fs/2 ,   (alternative is boxcar(N+1) window)
figure(5);  title(' Lowpass FIR Filter  ')
freqz(0.99*h,1,512,L*fs);       % direct plot of abs(G) = abs(fft(b,n)./fft(a,n))
[G,f] =freqz(h,1,1024,'whole',L*fs);% w in Hz;  % whole : f up to Lfs
figure(6);
plot(f,abs(G)); hold on
plot(fup,MXUP(1:length(XUP))); hold off;
grid;
axis([0 fs*L 0 1.01])
xlabel('f in Hz'), ylabel('|G(f)|     2|Xk|/(L*m)')
title('Magnitude: Lowpass Filter and Spectral Components of Upsampled Signal ')
```

Code 2-11: Interpolation lowpass filter suppresses all imaging spectral components, f_C = 10.625 kHz.

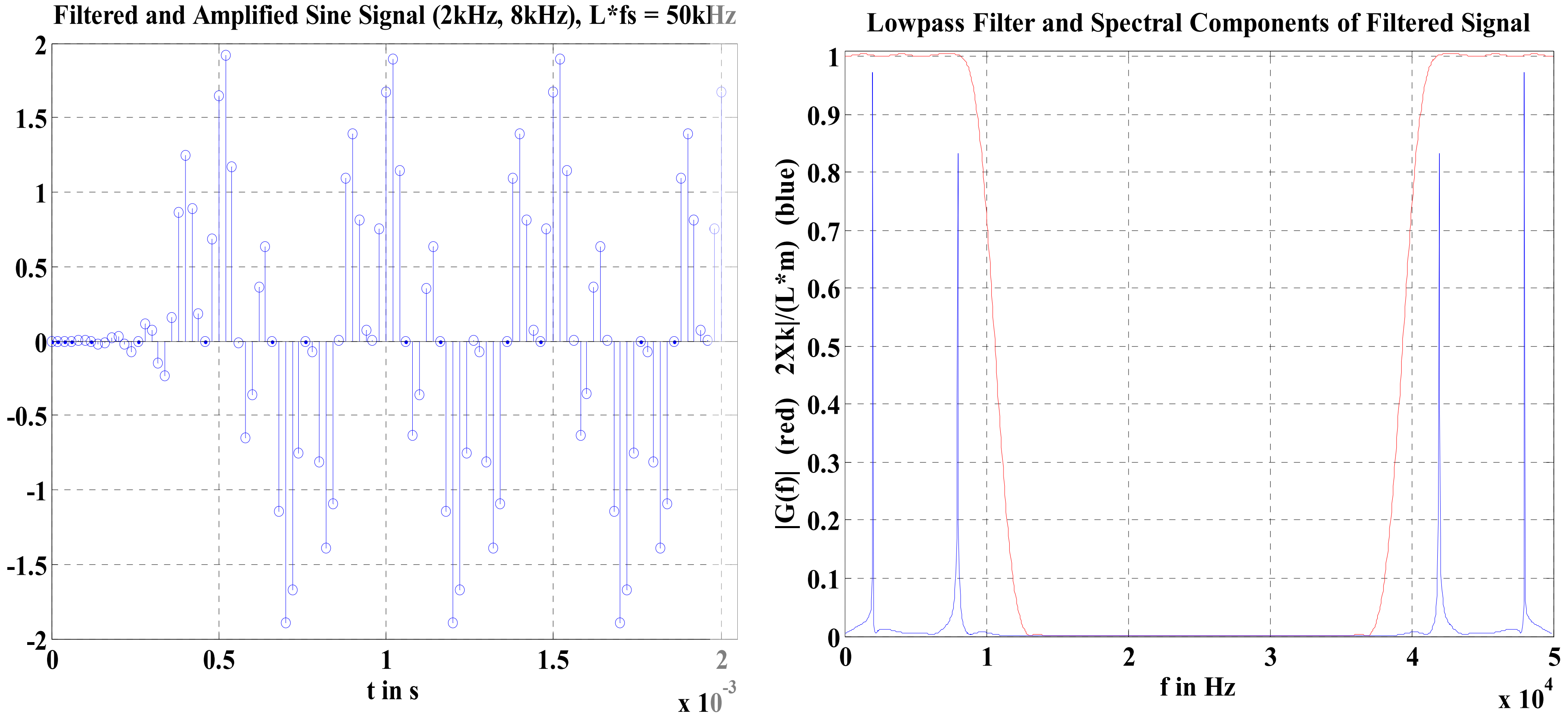


Fig. 2-24: Interpolator output signal with f_{Snew} = 50 kHz. Reconstruction of original spectral components.

```
% Elimination of image frequencies after interpolation
xf = 2*filter(h,1,xup);      % Amplification with gain L: compensation of averaging effect
tshort = 0:Ts/L:(floor(m/10)-1)*Ts; % (1. t value: Δ T: max t value)
figure(7);
stem(tshort,xf(1:floor(L*m/10)-1));
axis([0 m*Ts/10 -2 2]); grid;  %
title('Filtered and Amplified Sine Signal (2kHz, 8kHz), L*fs = 50kHz')
xlabel('t in s'), ylabel('    ')
XF = fft(xf,length(xf));
figure(8)
plot(f,abs(h)); hold on
plot(fup(1:L*m-1),2*abs(XF(1:L*m-1))/(L*m-1));hold off; grid
axis([0 fs*L 0 1.01])
xlabel('f in Hz'), ylabel('|G(f)|  and  2|Xk|/(L*m)')
title('Magnitude: Lowpass Filter and DFT of Filtered Signal')
```

Code 2-12: Lowpass filtering with amplitude correction(*L). Spectral components of upsampled and filtered signal.

2.5.3 Sampling Rate Conversion by a Rational Factor [5]

- In case that the input and output rate is not an integer factor, then a rational factor L/R has to be realised.
- An interpolator is used to increase the sampling rate by L in an input stage. A decimator is the next stage which decreases the sampling rate by 1/R.
- Both lowpass filters can be combined into one filter with the lower passband frequency and therefore take advantage from a reduced group delay.

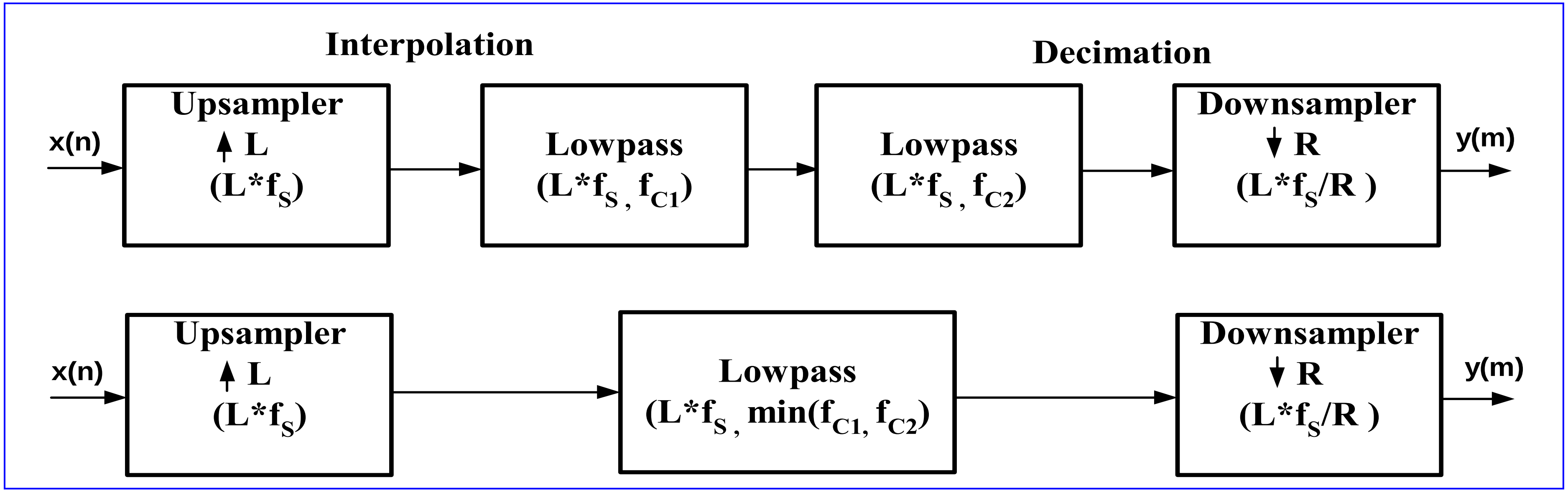


Fig. 2-25: Noninteger decimation respectively interpolation cascade.

2.5.4 Polyphase Decomposition of Sample Rate Conversion Filters

- **From the preceding chapter (comp. Fig. 2-25) we get that in both sample rate conversion blocks a lowpass is operating with the higher frequency (interpolator L*f_S, decimator f_S).**

- **Therefore the requirements which specify the timing of the digital signal processing system are driven by harder constraints.**

- **Investigations on reordering the sequence of filtering and sample rate conversion are highly motivated because we always should try to relax the constraints on digital hardware:**

  Reduce hardware costs or increase efficiency with given technology.

- **The hardware implementation of a decimator can be simplified because only every R output samples are of interested. Therefore the number of output calculation decreases and less pairs of coefficients and samples are necessary.**

- **Interpolation introduces additional L-1 zero value samples in between two original samples. It can be expected that even with increased sample rate no proportional increase of multiplications will result.**

Hardware Implementation of a Decimator

$$w(n) = \sum_{k=0}^{N} h(k)x(n-k) \qquad y(m) = w(Rm) = \sum_{k=0}^{N} h(k)x(Rm-k) \tag{2.5}$$

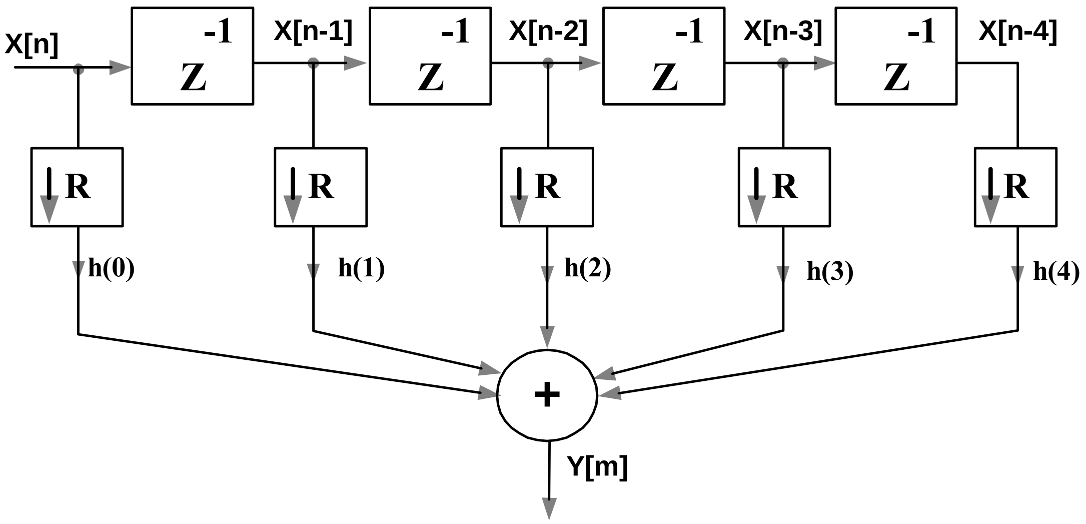


Fig. 2-26: FIR downsampler with reduced update rate at multipliers.

- Read new input x(n)
- Update delay line
- R samples obtained
- Compute y(m)

- For every R samples of x(n) applied to the delay line one output y(m) is computed.
- It is unnecessary to perform equation 2.5 for those samples w(n) that are discarded.
  - ➢ The down sampling operation can be performed before the multiplication.

- Example: N = 4 delay taps, N+1 coeffcients, R = 3, R-1 = 2 filter results are discarded

$y(0) = w(0) = h_0x_0$

$w(1) = h_0x_1 + h_1x_0$

$w(2) = h_0x_2+ h_1x_1 + h_2x_0$

$y(1) = w(3) = h_0x_3+ h_1x_2 + h_2x_1 + h_3x_0$

2.6

$y(2) = w(6) = h_0x_6+ h_1x_5 + h_2x_4 + h_3x_3 + h_4x_2$

$y(3) = w(9) = h_0x_9+ h_1x_8+ h_2x_7 + h_3x_6 + h_4x_5$

➢ x_0 only needs to be multiplied with h_0, h_R, h_{2R}, …

➢ These coefficients only need to be multiplied with x_0, x_3, x_6, x_9, x_{12}, …

- It is therefore reasonable to split the input signal x(n) first into R separate sequences and also the filter h(n) into R sequences:

$$x(n) = \sum_{r=0}^{R-1} x_r(n) \qquad h(n) = \sum_{r=0}^{R-1} h_r(n) \qquad 2.7$$

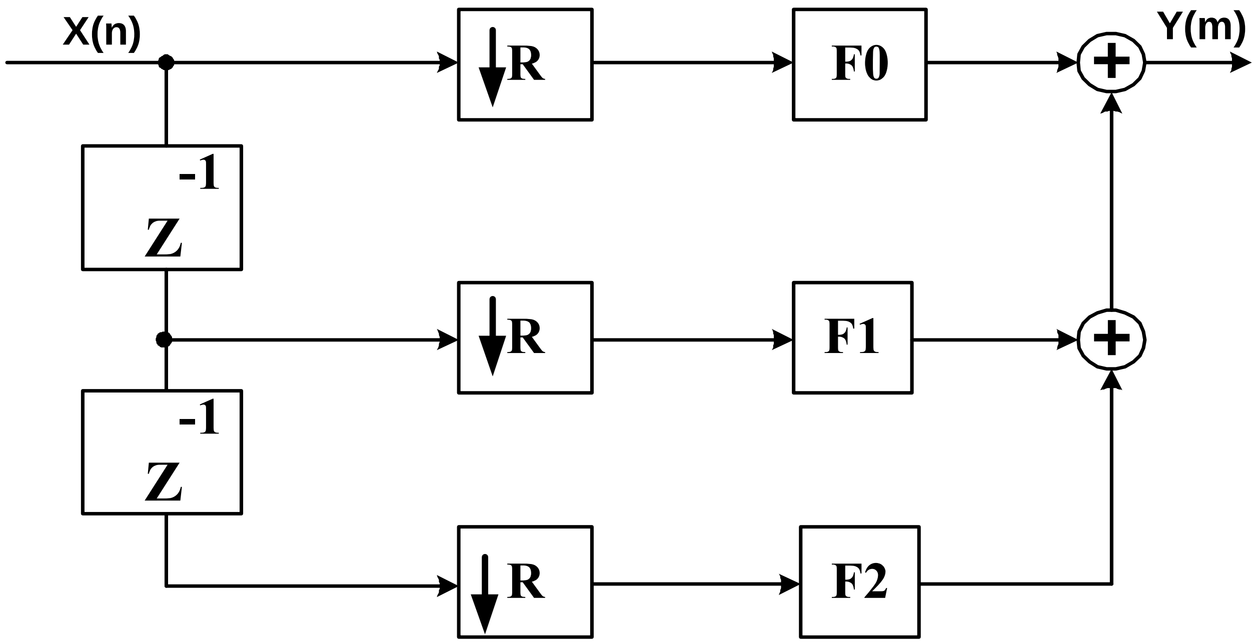


Fig. 2-27: Polyphase structure of a FIR decimation filter with R = 3.

- Filter arithmetic is performed with reduced frequency f_S/R.
- Such a decimator can run R times faster then the usual FIR filter followed by a downsampler.

- Filters F_i are called polyphase filters, because they all have the same magnitude transfer function, but they are separated by a sample delay, which introduces a phase offset.
- Example: R = 3, N = 10 , N+1 coefficients calculated for f_S, $f_C < f_s/(2M)$; operating frequency f_S/R:

$$F_0 = h_0 + h_3z^{-1} + h_6z^{-2} + h_9z^{-3}$$

$$F_1 = h_1 + h_4z^{-1} + h_7z^{-2} + h_{10}z^{-3} \qquad 2.8$$

$$F_2 = h_2 + h_5z^{-1} + h_8z^{-2}$$

Down sampling realised with:

- Counter running with f_S which enables the F_i delay stages every R's f_S cycle.

- Xilinx DLLs provide clock deviders and phase shifted outputs.
  - In *Fig. 2-28* a decimator input stage for R = 2 is depicted with

    a handover synchronisation register (reg 1, f_S),

    a polyphase decomposition input delay stage (reg 2, f_s)

    and two handover registers (reg3, reg4, f_S/R) which realise the down sampling.
  - *Fig. 2-29* depicts the cascade of two DLLs: The first provides the clock devider f_S/R and the second introduces a phase shift of 90^0 to the devided clock.

    With this phase shift it is ensured that the appropriate sample is introduced to the F_i chain "shortly" after it is available at the synchronisation register reg1 or the delay stage reg2.
  - The timing of a input sample stream is presented with *Fig. 2-30*.

    The first nibble value is F in order to recognise that F_0 starts with F and F_1 starts with 0 from the reset of delay stage reg2.

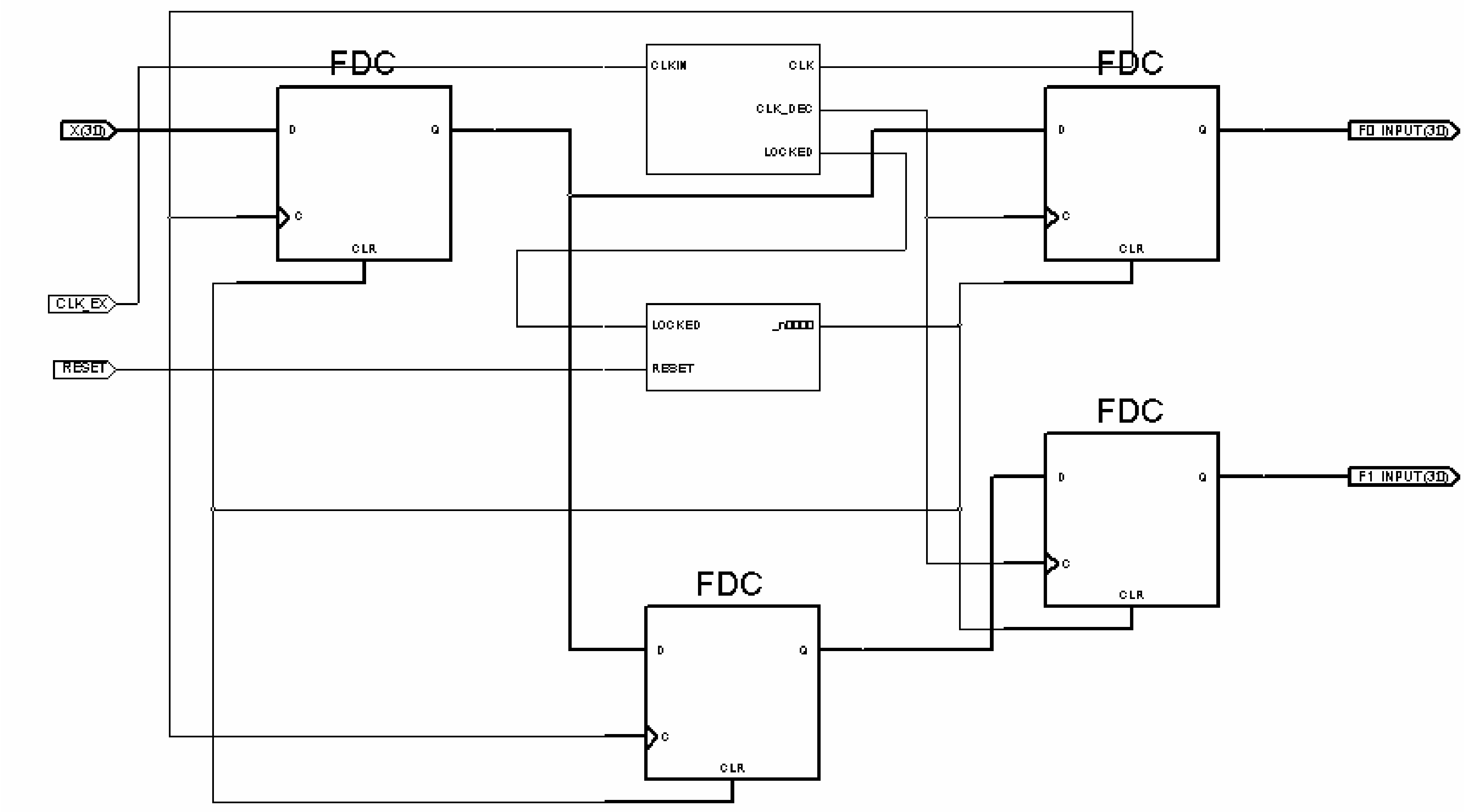


Fig. 2-28: Input stage to a decimator R = 2 for polyphase decompositon. Clock domain change.

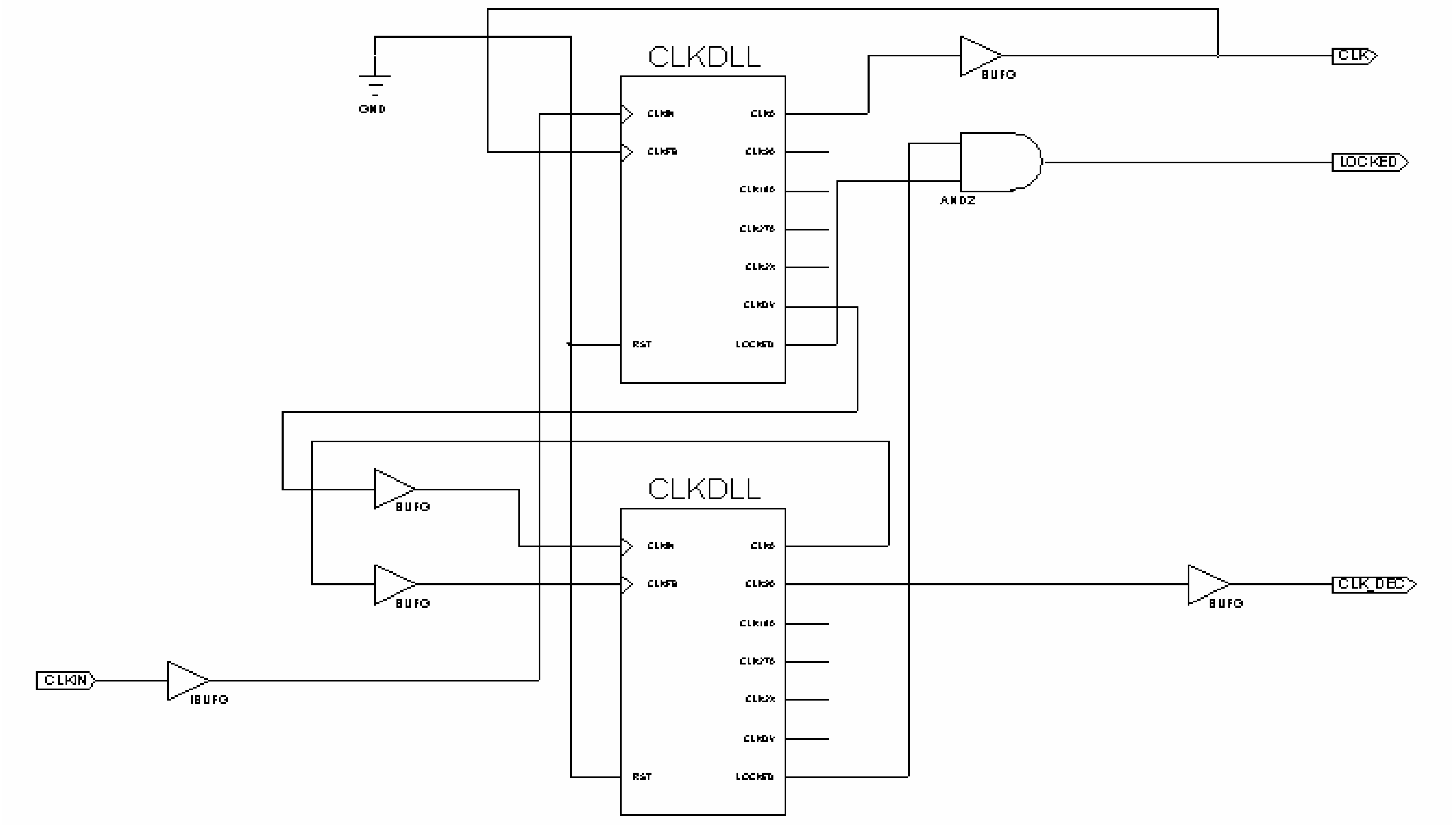


Fig. 2-29: DDL cascade for the clock supply to a decimator.

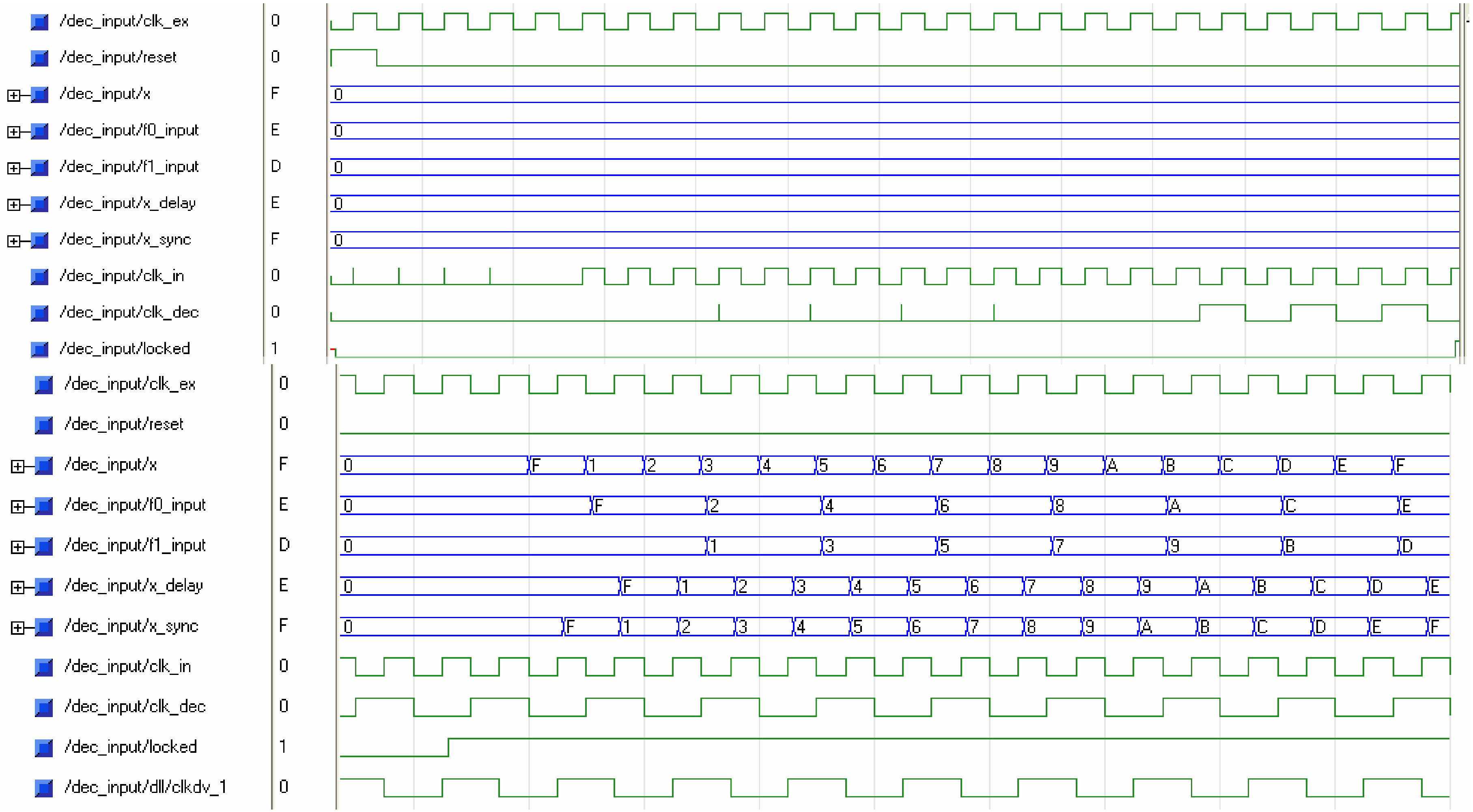
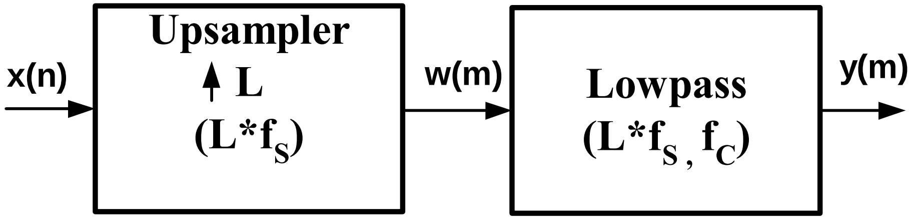


Fig. 2-30: Filling the decimator(R = 2) input stage with a stream of nibbles (F,1, 2, 3, …, E, F). Two clocks have to be established CLK_in and CLK_dec.

Hardware Implementation of an Interpolator

Interpolation



x(n)

Upsampler

↑ L

(L*f_S)

w(m)

Lowpass

(L*f_S, f_C)

y(m)

$$y(m) = \sum_{k=0}^{N} h(k)w(m-k)$$

$$\mathrm{w(m-k)} = \begin{cases} \mathrm{x((m-k)/L)}, & \mathrm{m-k = 0, L, 2L, 3L, ..} \\ 0, & \mathrm{m-k = 1, ..., L-1, L+1, ..., 2L-1, 2L+1} \\ & \mathrm{, ..., 3L-1, 3L+1, ...,} \end{cases} \qquad 2.9$$

- **The lowpass filter is filled with L-1 zeros after an updated input sample x(n) was introduced.**
- **Because several products will be zero the filter’s convolution sum does not depend on every delay stage output at each clock f_S cycle.**

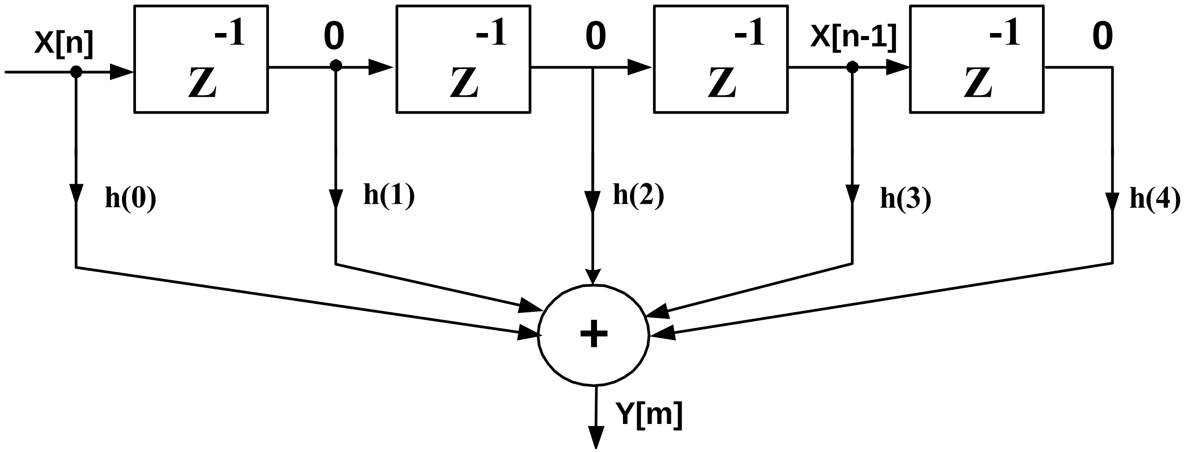


Fig. 2-31: Filter with upsampled inputs w(m) (N = 4, L =3).

- **Example:**

N = 6 delay taps,

N+1 coeffcients $h_0 – h_6$,

L = 3, L-1 = 2 zeros are filled in

$$y(0) = h_0w_0 = h_0x_0$$
$$y(1) = h_0w_1 + h_1w_0 = 0 + h_1x_0$$
$$y(2) = h_0w_2 + h_1w_1 + h_2w_0 \;= 0 + 0 + h_2x_0$$

2.10

$$y(3) = h_0x_1 + 0 + 0 + h_3x_0$$
$$y(4) = 0 + h_1x_1 + 0 + 0 + h_4x_0$$
$$y(5) = 0 + 0 + h_2x_1 + 0 + 0 + h_5x_0$$

$$y(6) = h_0x_2 + 0 + 0 + h_3x_1 +0 + 0 + h_6x_0$$
$$y(7) = 0 + h_1x_2 + 0 + 0 + h_4x_1$$
$$y(8) = 0 + 0 + h_2x_2 + 0 + 0 + h_5x_1$$

- This example expresses that three (L) kind of sums have to be calculated each with different coefficients operating on a maximum of three input samples: one up to date sample x(n) and one respectively two previously stored samples x(n-1) , x(n-2).

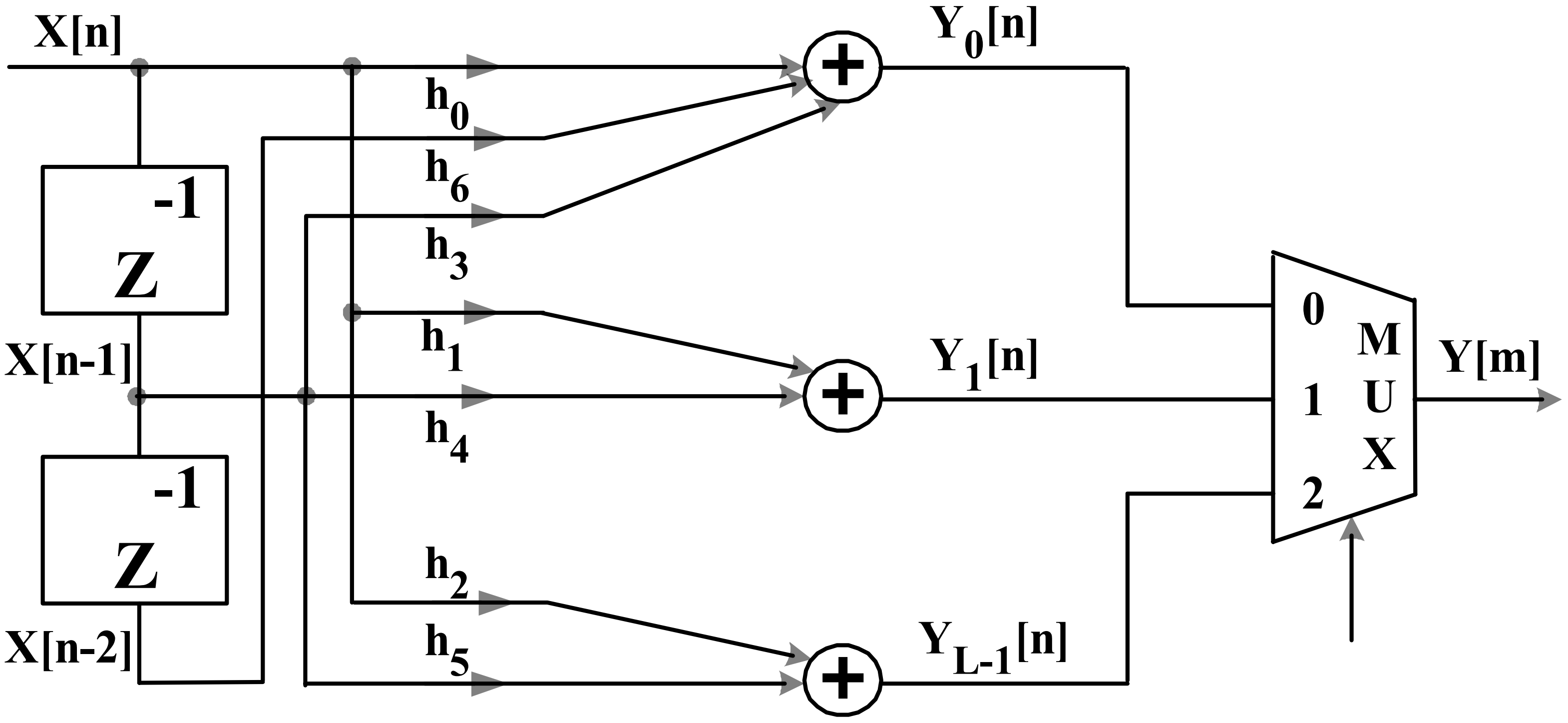


Fig. 2-32: Interpolator lowpass filter with N = 6, L = 3, N+1 coefficients calculated for Lf_S, $f_C < f_s/2$, operating frequency f_S. Reduced number of delay stages floor(N/L) and upsampling output multiplexer.

- **Each triple set of equations (2.10) is updated with the same input sample x(n). In order to provide all needed samples only two (floor(N/L)) delay stages are necessary which are clocked by f_S.**

- **Based on the floor(N/L) delay stages three simplified FIR filters of maximum order N/L contribute to the interpolator output y(m) (N = 6, L = 3):**

$$F_0 = h_0 + h_3 z^{-1} + h_6 z^{-2}$$

$$F_1 = h_1 + h_4 z^{-1}$$

$|F_1| = |F_2|$ equal magnitude responses!

$$F_{L-1} = h_2 + h_5 z^{-1}$$

- **The partial sums $y_i(n)$ (i = 0, 1, …, L-1) (comp. Fig. 2-32) are upsampled by a multiplexer which mixes the output y(m) by connecting it to the $y_i(n)$ in a dedicated sequence:**

$y_0(n)$ $y_1(n)$ y_2 (n) $y_0(n)$ $y_1(n)$ $y_2(n)$ … All these elements have a duration of $1/(Lf_S)$

- **This sequence is established by the multiplexer's select input which is driven by a modulo L counter clocked by Lf_S.**
- **The longest combinational delay path which begins at the delay stage output and ends at the multiplexer's output can be broken by pipelining registers. Registers can be inserted in between the multipliers and the adders and the multiplexer as well.**