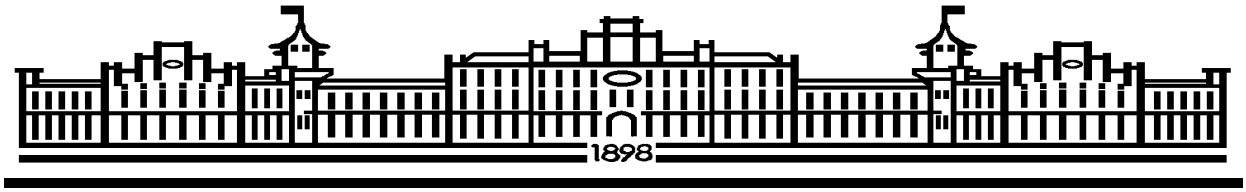


МІНІСТЕРСТВО ОСВІТИ ТА НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ ІМЕНІ ІГОРЯ СІКОРСЬКОГО»

Методичні вказівки до виконання лабораторних робіт

по курсу

Технологія проектування комп'ютерних систем — 2

(назва дисципліни)

для напряму підготовки (спеціальностей)

123 Комп'ютерна інженерія

(шифр та назва напряму, спеціальностей)

(in English)

Methodical instructions for laboratory excercises

in the course

Computer Systems Design Technology – 2

Уклав
професор Анатолій Михайлович
Сергієнко, д.т.н., с.н.с. кафедри ОТ

Рекомендовано
Вченою радою факультету
інформатики та обчислювальної техніки
НТУУ «КПІ ім. Ігоря Сікорського»
Протокол № 11_ від _7_.__05_____.2019 р.

Київ - 2020

Laboratory Excercise 1

Recursive digital filter design

Goal: To gain the knowledge and skills in the development and testing of the high-speed digital filters in FPGA.

Theoretical information

Transfer function

Recursive, or infinite impulse response (IIR) digital filters are often used in digital signal processing. These filters being configured in the field programmable gate arrays (FPGAs) have high speed and low energy consumption. Any IIR filter is described by a N-th order difference equation with the constant coefficients:

$$y(n) = -\sum_{k=1}^{N} a_k y(n-k) + \sum_{r=0}^{M} b_r x(n-r). \quad (1)$$

Thus, the n-th value of the output can be calculated on the basis of the n-th value of the input x (n) and, accordingly, of N and M past values of the output $y(n–k)$ and the input $x(n–r)$. Then the ***impulse response*** of such a system can be defined as:

$$h(n) = \frac{y(n)}{x(n)} \ . \quad (2)$$

The filtering of the signal x by the filter impulse response h is called the ***convolution***. The impulse response (2) in the general case is infinite and then it is an IIR system.

The convolution of $h(n)*x(n)$ corresponds to the multiplication $H(z)X(z)$ in the z-space. It should be noted that these convolution properties are valid only for the domain of the complex variable z, where the function does not diverge.

The impulse response $h(n)$ of system (2) is mapped in the z-space as a ***transfer function***:

$$H(z) = \frac{Y(z)}{X(z)} = \frac{\sum_{r=0}^{M} b_r z^{-r}}{1 + \sum_{k=0}^{N} a_k z^{-k}}, \quad (3)$$

where a_k, b_r are real numbers, the function z^{-m} corresponds to the delay of the signal to m cycles of the sampling clock signal. Here, the number N is a system order.

The transfer function (3) can be decomposed into the sum of elementary fractions. Most often, for example, if $N \geq M$; $Q = N/2$, it is factored into the product of fractions:

$$H(z) = \prod_{k=0}^{Q} \frac{b_{0,k} + b_{1,k} z^{-1} + b_{2,k} z^{-2}}{1 + a_{1,k} z^{-1} + a_{2,k} z^{-2}} \quad . \quad (4)$$

The transfer function indicates the spectral properties of the linear system under consideration. On its basis, we find the ***magnitude-frequency characteristic*** $|H(e^{-j\omega t})|$ and the ***phase-frequency characteristic*** $\arg(H(e^{-j\omega t}))$ of the system.

Graphical representation of the IIR filter algorithm

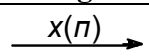
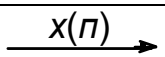
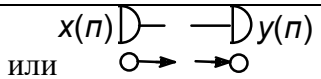
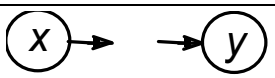
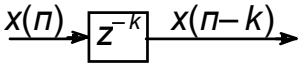
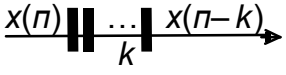
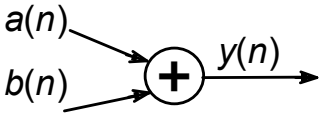
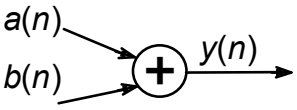
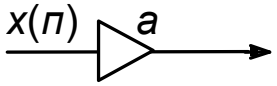
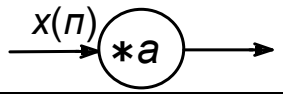
In the vast majority of DSP algorithms, epy signal flows are synchronous. Therefore, such algorithms can be represented by a synchronous data flow graph (SDF). The signal flow graph is commonly used in the DSP algorithm considering. And such an algorithm is equivalent to homogeneous SDF. Table 1 shows the correspondence of the graphical notations of the elements of both signal flow graph and SDF.

Consider an example of the IIR filtering algorithm of a high-pass second-order filter (HPF) using a signal graph and SDF. The transfer characteristic of such a filter is equal to

$$H(z) = \frac{1 - z^{-2}}{1 + a_1 z^{-1} + a_2 z^{-2}} = (1 - z^{-2}) \cdot \frac{1}{1 + a_1 z^{-1} + a_2 z^{-2}}, \quad (5)$$

(compare with (4)).

Table 1. Designation of the elements of the signal graph and GSPD

Model element	Signal flow graph	Uniform SDF
Signal $x(n)$		
Input, output ports, signal source $x(n)$, and destination $y(n)$		
Delay to k cycles		
Signal addition, adder node $y(n) = a(n)+b(n)$		
Multiplication to a constant $y(n) = a \cdot x(n)$, multiplication node		

The differential equations correspond to the factors of the transfer function of the filter (compare with (1)):

$$u(n) = x(n) - x(n-2);$$

$$y(n) = u(n) - a_1 y(n-1) - a_2 y(n-2).$$

The equations are calculated in the signal flow graph (Fig. 1). All delay elements are considered to be zeroed before the algorithm execution. The input datum $x(n)$ is sampled with the sampling frequency f_S. As soon as $x(n)$ arrives, it immediately goes to the delay element for two cycles z^{-2} and to the adder "+", where it is added to the delayed data $x(n-2)$. The other elements of the graph model function in the same way: as soon as there is input datum for a node, it immediately triggers and outputs the output datum.

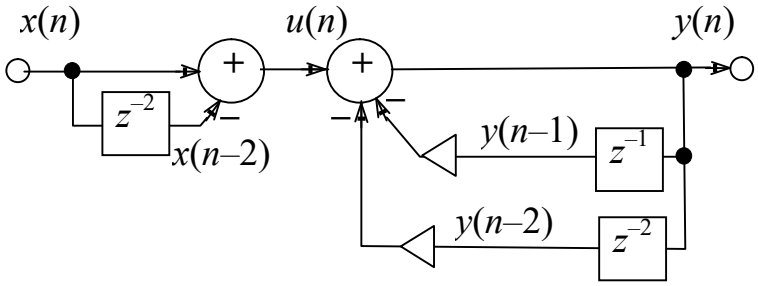
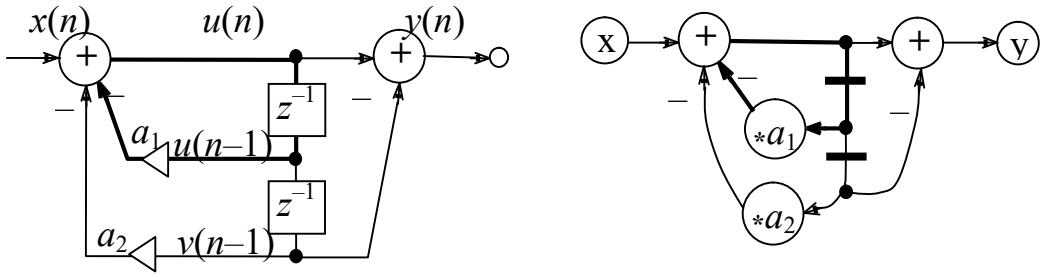


Рис. 1. Signal flow graph of a LPF algorithm

The graph in Fig. 1 can be considered as a structural diagram of some application-specific calculator, which adders, multipliers are derived by the mapping of addition, multiplication nodes, and registers are done by the mapping of delays. This variant of the algorithm is not a rational one because of the excessive number of delays. To optimize it, it is possible to rearrange the factors in formula (5) and to represent the delay z^{-2} as two consecutive delays z^{-1}:

$u(n)= x(n) – a_1u(n–1) – a_2v(n–1);$
$v(n) =u(n–1);$
$y(n) = u(n) – v(n–1);$

The resulting signal graph is shown in Fig. 2, a. It corresponds to SDF in Fig. 2, b. This graph is called the canonical form of an IIR filter, since it contains a minimum number of delay elements for storing the intermediate results, which are the samples of delayed signals $u(n)$ and $v(n)$.



a) b)

Fig. 2. Signal flow graph of HPF, a) and respective SDF, b)

The signal flow graph and respective SDF may have closed cycles. In Fig. 2 such a cycle is highlighted by a thick line. If there is no delay elements in the closed loop, then the signals in it are endlessly re-assigned within one clock cycle, that is, the algorithm is blocked. Therefore, at least one delay element in each closed loop is a prerequisite for the absence of deadlocks in the signal flow graph or SDF. Another condition for the absence of the deadlocks is the initial data, for example, zero data in all the delay elements that belong to such cycles.

IIR filters based on phase filters

The phase filter has a magnitude of its transfer function $H(z)$ which is equal to $|H(z)| = 1$. Its phase frequency response at a frequency f_R has a phase shift equal to 180°. If the signals from two parallel phase filters are added, the output signal is suppressed at the frequencies for which the phase difference is 180°. The resulting transfer function is:

$$H_S = (H_1(z) \pm H_2(z))/2, \tag{6}$$

and it corresponds to various filters: low pass filter (LPF), high pass filter (HPF), bandpass filter (BPF), or notch filter. The filter orders play the role as well. For example, if the function $H_1(z)$ is of the second order, as in this laboratory work, then for $H_2(z) = 1$ we obtain a notch filter, for $H_2(z) = -1$ we do BPF, for $H_2(z) = \pm z^{-1}$ we do LPF (+), and HPF (–).

The phase-based IIR filter is characterized by the stability at the low bit rate of its coefficients, high linearity of the frequency response, as well as high speed. The parameters of its frequency response, such as the position of the cutoff frequency, the slope of the transition band are directly dependent on the coefficients of the filter.

The LPF transfer function, which can be reconfigured, is:

$$H_S = (H_1(z) + z^{-1})/2,$$

where
$$H_1(z) = \frac{b + a(b+1)z^{-1} + z^{-2}}{1 + a(b+1)z^{-1} + bz^{-2}}, \tag{7}$$

$$a = \cos(2\pi f_R),$$

$$b = (1-t)/(1+t), \tag{8}$$

$$t = \mathrm{tg}(\pi \Delta f),$$

moreover, the coefficient a regulates the cutoff frequency f_R, coefficient b sets the width of the transition band Δf or the ***cutoff sharpness***. Thus, changing a in (7), the ***edge frequency*** of the passband is regulated within $(0.1 - 0.4)f_S$ with a suppression in the stop band up to 50 dB.

SDF of LPF and SDF of the bandpass filter, which are constructed in accordance with (6), are shown in Fig. 3. Therefore, these SDFs are distinguished only by the sign of addition and the presence or absence of a delay in the second branch of the input signal propagation.

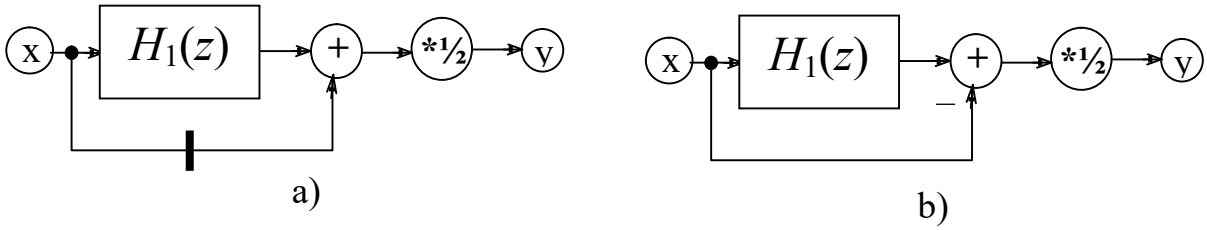


Fig. 3. SDF of LPF, (a) and bandpass filter SDF, (b)

The bandpass filter works as follows. The same signal passes through two branches of the graph and a copy of it is subtracted at all frequencies except the resonance one, giving a zero result. At the resonant frequency f_R, the phase filter returns the signal by 180° and as a result, the signal and its copy are added. LPF works similarly, but the signal in the phase filter returns 180° at frequencies above f_R.

Let us consider in detail several examples of SDF of the second-order phase filter. In the direct implementation of the formula (7), at least 6 multiplication nodes and 6 adder nodes are required in SDF. There is a more efficient SDF for this formula, which is called the wave-propagation filter graph because it is a waveguide model (Fig. 4).

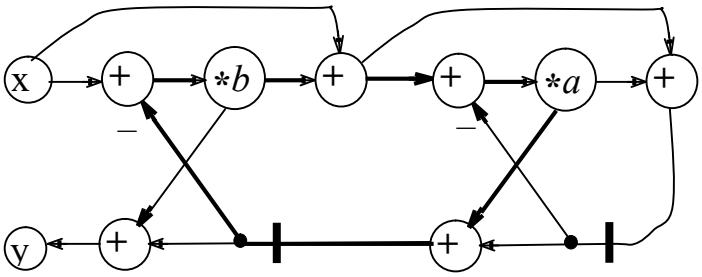


Figure 4. SDF for the transfer function (7)

Formula (7) can be reduced by rejecting the ability to directly control the characteristics of the filter:

$$H_{12}(z) = \frac{b + cz^{-1} + z^{-2}}{1 + cz^{-1} + bz^{-2}}, \tag{9}$$

where $c = a(b + 1)$. Respective SDF, which represents the cannonic filter structure, is shown in Fig. 5. The ***cannonic*** filter structure contains the minimum delay number which is equal to the filter order.

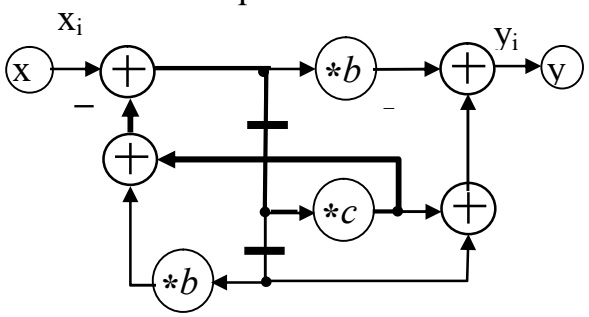


Figure 5. SDF for the transfer function (9) in the canonical form

Formula (9) corresponds to the difference equation (10).

$$y_i = b \cdot x_i + c \cdot x_{i-1} + x_{i-2} - c \cdot y_{i-1} - b \cdot y_{i-2} \ . \tag{10}$$

Equation (10) can be rewritten as follows

$$q_i = x_i - c \cdot q_{i-1} - b \cdot q_{i-2} \ ; \tag{11}$$

$$y_i = b \cdot q_i + c \cdot q_{i-1} + q_{i-2}.$$

If the delays in edges are not to be minimized, then equations (11) are satisfied in SDF, such as in Fig. 6. But we get only two multiplication operations. When selecting the common coefficients for parentheses in (10), we obtain the equation:

$$y_i = b \cdot (x_i - y_{i-2}) + c \cdot (x_{i-1} - y_{i-1}) + x_{i-2} \ . \tag{12}$$

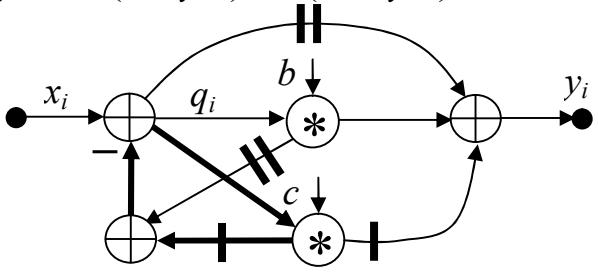


Figure 6. SDF for the transfer function (9) with excessive delay number

According to the equation (12) SDF is drawn, which is shown in Fig. 7. Due to the reuse of delays, the authors Mitra and Hirano had constructed the SDF example called MH2B, which is shown in Fig. 8.

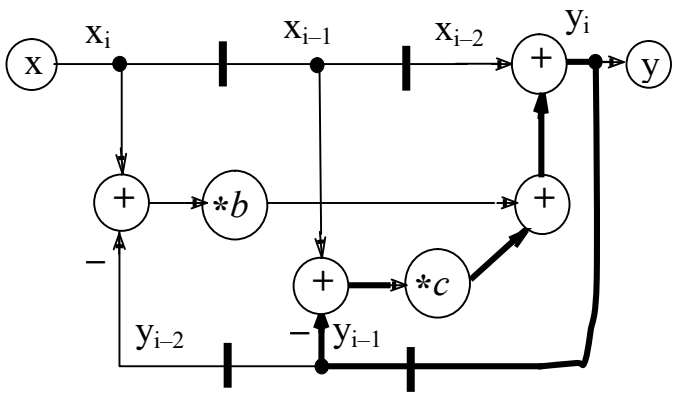


Fig.7. SDF for equation (12)

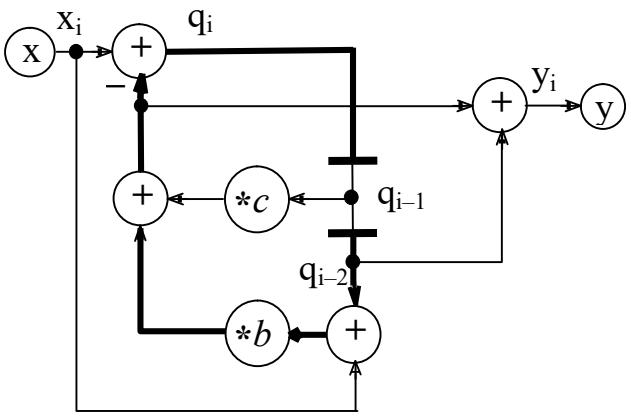


Fig.8. SDF MH2B

The thick line in SDF in Fig. 4 – 8 represents the critical path. The length of this path has the maximum value — $4t_S + 2t_M$ for SDF in fig. 4 and the minimum value for SDF in Fig. 5 and 6, which is equal to $2t_S + t_M$ at the cost of the larger number of multiplication nodes.

Digital filter testing

Determination of the frequency response of a filter is a typical procedure for its diagnosis and testing. For the analysis or measurement of the magnitude-frequency response and the phase-frequency response of a IIR filter, a complex signal $e^{-j\omega n}$, which, should be fed to the input of the tested filter. Here, $\omega = 2\pi f$ is the frequency under consideration. This signal is called an ***analytical signal***.

Often, a simple method of checking is used which uses only the component $\cos(\omega n)$ instead of the analytical signal. Then, the frequency response is measured at the output of the system, as a maximum of the resulting signal $Re(H(\omega))$, that is, it is measured at the moments when the second component is equal to $Im(H(\omega)) = \sin(\omega n) = 0$. The disadvantage of this method is the inaccuracy of measuring the maximum of the signal.

A more accurate method is based on deriving the imaginary component $Im(H(\omega))$ after passing the result of $Re(H(\omega))$ through a Gilbert filter, which rotates the phase of the signal by 90°. But such a filter introduces a significant distortion in the frequency response.

Therefore, a signal graph such as in Fig. 9 should be used to analyze the IIR filter. It uses two identical copies of the filter with the function $H(z)$ to which the sine and cosine components of the analytical signal are fed. Respectively, the components of the analytic response signal of the IIR filters are measured on their outputs.

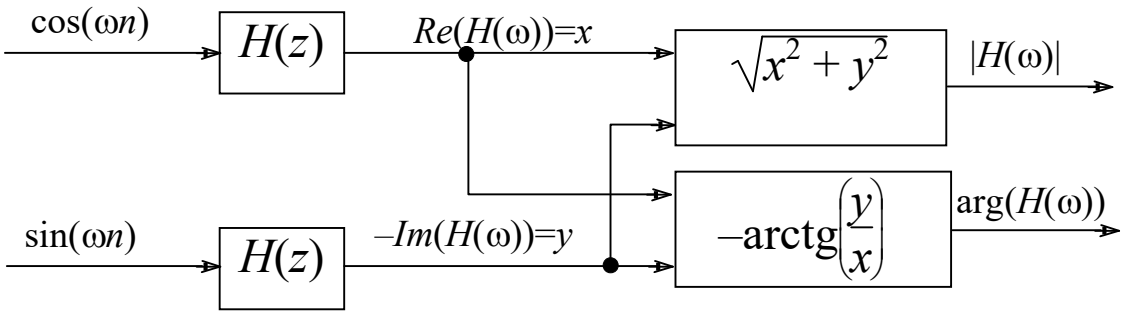


Fig. 9. Frequency response measurement of the real system $H(z)$

In the laboratory work, a test bench is used, having the structure as in Fig. 10, which can be downloaded at: http://kanyevsky.kpi.ua/en/useful-ip-cores/testbench-for-the-filter-testing/ The ports and tuning constants of this testbench unit are presented in Table 1.

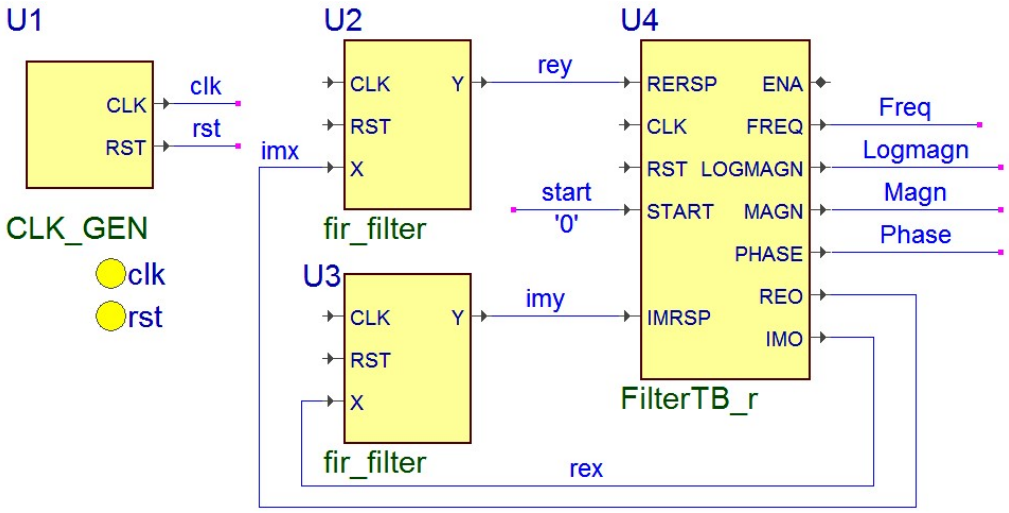


Fig.10. Filter testbench

Filter bit width selecting

The vast majority of IIR filters are calculated in computers or FPGAs using the integers or fixed-point numbers. When programming such a filter, a set of coefficients that meet the requirements are presented in a floating-point format. Then, the numbers of bits of quantization of the coefficients $n_к$, the input data $n_{x,}$ and the results n_y are chosen.

As a rule, $n_x, n_y \geq \log_2 D$, where D is the dynamic range of the signal. That is, every 6 decibels of the dynamic range account for at least one bit of data.

The coefficients are scaled and rounded off, so that at $|b_i| < 1$ the integer value is equal

$$b'_i =]2^{nx} \cdot b_i + 0.5[\ . \qquad (11)$$

Table 1. Ports and generic variables of the testbench module for testing the digital filters

Port name	**Port meanings**
fsampl	Integer sampling frequency, for example, 1000 kHz
fstrt	Starting frequency fo, the first frequency which is analyzed
deltaf	Frequency increase d, so in k steps the generator will output the frequency fo + k*d
maxdelay	The delay in signal samples, after which the output signal parameters will be estimated. Usually it is slightly higher than the maximum (group) delay of the filter which is tested.
slowdown	Factor, in which the filter speed is slowed down. If the input samples enter each clock cycle then slowdown=1, if the samples go in odd clock cycles then slowdown=2, etc.
nn	Input and output data width
magnitude	Integer magnitude of the generated sine/cosine waves. For example, if nn=8 then magnitude is any positive number less than 127.
REO,IMO	Cosine/ sine waves, represented by the nn bit integers, outputted by the component
RERSP,IMRSP	Filter output signals , which are responses to the cosine/ sine waves, and which must be ported to the nn-bit width inputs
FREQ	Frequency code of the given sine/cosine waves in this but the previous frequency step, which is equal to fo + (k-1)*d
MAGN	Estimated magnitude of the signal RERSP,IMRSP at the frequency FREQ
LOGMAGN	Estimated magnitude of the signal RERSP,IMRSP at the frequency FREQ in the logarithmical scale, i.e. in decibels. Note, the signal with the given magnitude is 0 db
PHASE	Estimated phase of the signal RERSP,IMRSP at the frequency FREQ represented in the range ± π
ENA	Enable signal, which strobes the filter inputs when slowdown>1

The filter results are calculated by the formula (1). In this case, the adder accumulating the result should have such a bit width so that no overflow occurs. Moreover, the bit width of the product is $n_д = n_к + n_х$, and the bit of the adder must be at least $n_с = \log_2 S + n_к + n_х$, where S is the theoretically possible maximum result of the formula (1). For the right half of the formula (1), which corresponds to the FIR filter, the maximum of the sum is equal to the sum of the modules of all the coefficients of the filter contained in the numerator of the formula of the transfer function, i.e.

$$n_c = \log_2 \sum_{i=0}^{M} |b_i| + n_к + n_х. \quad (12)$$

For the left half of the formula (1), the maximum of the sum can be much larger due to the amplification of the signal in the feedback. Such gain is proportional to the filter quality factor, which is proportional to the gain of the signal at the resonant frequency. For the phase filter (7) and (8), the quality factor of the filter increases sharply at $b \to 1$.

In practice, for the phase filters and filters based on them, due to their low sensitivity to rounding errors, the bit of intermediate results is chosen as

$$n_c = n_д + n_х + 3, \quad (13)$$

where $n_д = 1,...,6$ is chosen depending on the filter quality factor and is specified after its simulation. That is, this value is reduced to such a minimum, which provides the overflow absence for all inner signals. For example, when $b < 0.7$ in (7), then it is enough $n_д = 1$.

The coefficient bit width for the phase filters may be less than one for other IIR filters. Usually. the bit width $n_к = n_х$ is sufficient. This bit width can be reduced according to the results of the filter testing if the resulting frequency response requirements are satisfied.

The filter result $y(n)$ is taken as the highest n_y bits of the sum (1), and the lower bits of the sum are truncated. Some other rounding algorithms can be used as well.

Since n_c can be quite a large number, and the probability of reaching the result of the maximum value is small (at resonant frequencies and large input signals), in practice, n_c is chosen slightly less. But in this situation, the addition in (1) is performed by the algorithm of accumulation with saturation. According to this algorithm, if there is an overflow occurs, the result is replaced by the maximum number with the corresponding sign. The signal saturation simulates a similar process in the analog circuits and produces significantly less signal distortion than the overflow.

The following example shows the implementation of the saturation in VHDL. Consider $n_с = 14$, $n_д = 3$ и $n_у = 8$, and the resulting magnitude is less than 1, and a fixed point stands before the 8-th bit. Then the saturation operation of the accumulated result S is programmed as

```
Y <= x”7F” when S(13 downto 10) > signed(“0001”),
     X”80” when S(13 downto 10) < signed(“1111”), else
     S(11 downto 3);
```

Here, the constants x”7F” and x”80” represent the maximum value 0,99 and minimum value –0,99.

Task for work

Develop a VHDL-project of a digital filter with a transfer function (6).

The types of components $H_1(z)$ and $H_2(z)$ are set according to the variant from Table 2. The task number of this and other laboratory works coincides with the student number in the group list. In this case, in Table 2, $H_2(z)$, cut-off frequency (resonance frequency) f_R, transition bandwidth Δf, and type of SDF are set. The coefficients *a, b* for the function (7) are calculated by formulas (8), and the coefficient *c* for the function (9) is calculated as $c = a\,(b + 1)$.

The bit width of the input and output data

in the first group, bit width is 14,

in the second – bit width is 16,

in the third – bit width is 24,

in the fourth – bit width is 18.

The coefficient bit width is equal to $n_к = n_x - 2$. The bit width of the internal intermediate results is determined by the relation (13).

The filter model is a description of a given SDF in VHDL. SDF is optimized by the retiming and pipelining methods. For example, it is advisable to add delays at the input and output of SDF, which are mapped in the corresponding registers of the input and output signals.

Table 2. Parameters and functions for laboratory work

var.№	f_R,	Δf	**Fig. SDF** $H_1(z)$	$H_2(z)$
1	0,1	0,05	4	z^{-1}
2	0,125	0,05	5	z^{-1}
3	0,15	0,05	6	z^{-1}
4	0,175	0,05	7	z^{-1}
5	0,20	0,05	8	z^{-1}
6	0,225	0,05	4	z^{-1}
7	0,25	0,05	5	z^{-1}
8	0,275	0,05	6	z^{-1}
9	0,1	0,05	7	$-z^{-1}$
10	0,125	0,05	8	$-z^{-1}$
11	0,15	0,05	4	$-z^{-1}$
12	0,175	0,05	5	$-z^{-1}$
13	0,20	0,05	6	$-z^{-1}$
14	0,225	0,05	7	$-z^{-1}$
15	0,25	0,05	8	$-z^{-1}$
16	0,275	0,05	4	$-z^{-1}$
17	0,1	0,1	6	1
18	0,125	0,1	7	1
19	0,15	0,1	8	1
20	0,175	0,1	4	1
21	0,20	0,1	5	1
22	0,225	0,1	6	1
23	0,25	0,1	7	1
24	0,275	0,1	8	1
25	0,125	0,1	4	–1
26	0,15	0,1	5	–1
27	0,175	0,1	6	–1
28	0,20	0,1	7	–1
29	0,225	0,1	8	–1
30	0,25	0,1	4	–1

The VHDL description of the filter should have appropriate comments that indicate the author and explain the execution of the algorithm.

The developed filter should be tested in the testbench, such as in fig. 8.

Also, the filter must be synthesized in FPGA CAD (Xilinx or Intel) for FPGAs selected arbitrarily, with the placement and routing procedures.

The laboratory work protocol should contain:

– filter algorithm, optimized filter SDF;

– VHDL-text description of the filter;

– magnitude-frequency response charts in a linear and logarithmic scale derived during the testing;

- synthesis results in the form of hardware costs and the minimum period of the clock frequency, given as a screenshot of the FPGA CAD.

Execution example

Consider SDF such as in Fig. 8 and it is necessary to develop a low-pass filter with a cutoff frequency f_R =0.25 (this is the so-called half-pass filter) and a transition band Δf = 0.1. The width of the input data is 12.

Then, according to (8) and (9), a = cos(2π·0,25) = 0; t = tg(π·0,1) =0,325; b = (1–0,325)/(1+0,325) = 0,509; c = a(1 + b) = 0. So, the multiplication to c is removed. The resulting filter SDF is shown in Fig. 11.

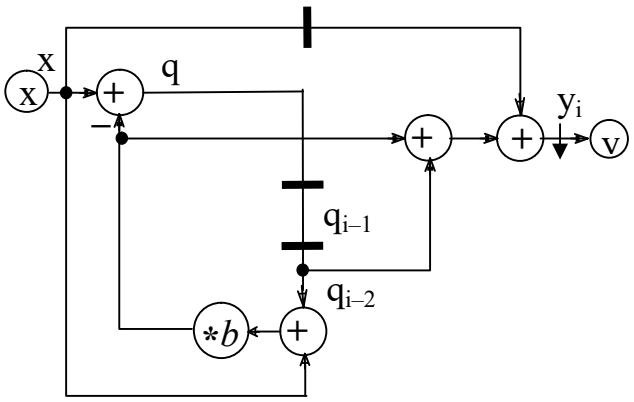


Fig. 11. Half-band low-pass filter SDF

In Fig.11, an arrow across the arc represents a one-bit right shift, that is, a division in a half.

Further, this SDF is subject to the retiming by pipelining. In this case, the graph edges are weighted with delays so that the algorithm remains unchanged, with the exception of latent delay and so that the critical path is minimized. The

resulting filter SDF is shown in Fig. 12. It shows the critical path and all the signals that are involved in the calculations.

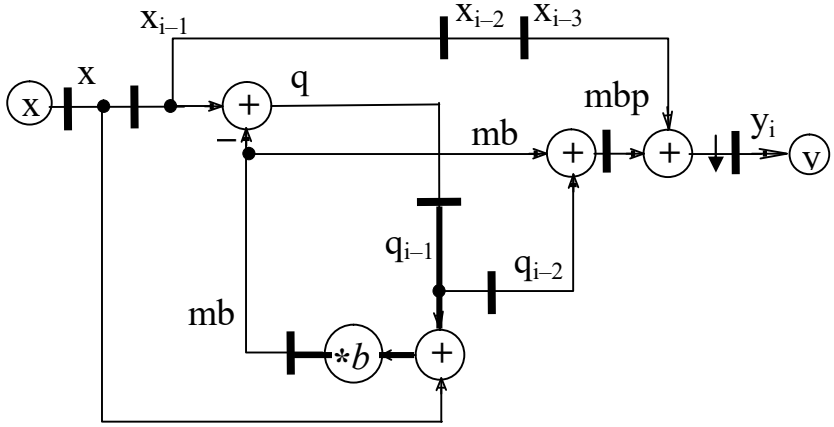


Fig. 12. Pipelined SDF of the half-band low-pass filter

The bit width of the intermediate signals $n_с = n_д + n_х + 3 = = 2+12+3 = 17$ is selected. The bit width of the coefficients is $n_к = 12$. The bit width of the product is $n_п = n_к + n_с = 17+12= 29$.

The resulting VHDL filter description is presented below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity LPF_HB_LAB is
     port(     CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          X : in STD_LOGIC_VECTOR(11 downto 0);
          Y : out STD_LOGIC_VECTOR(11 downto 0)
          );
end LPF_HB_LAB;

architecture synt of LPF_HB_LAB is
     constant b:signed(11 downto 0):=
           to_signed(integer(0.509*2.0**11),12);
     constant max:signed(3 downto 0):="0111";
     constant min:signed(3 downto 0):="1100";
     signal xi,xi_1,xi_2,xi_3:signed(16 downto 0);
     signal qi,qi_1,qi_2,mbp:signed(16 downto 0);
     signal mb:signed(28 downto 0);
     signal yi:signed(11 downto 0);
begin
     LPF:process(CLK,RST)
       variable yt:signed(16 downto 0);
     begin
          if RST = '1' then
               xi<=(others=>'0');
```

```
                xi_1<=(others=>'0');
                xi_2<=(others=>'0');
                xi_3<=(others=>'0');
                yi<=(others=>'0');
                qi_1<=(others=>'0');
                qi_2<=(others=>'0');
                mb<=(others=>'0');
                mbp<=(others=>'0');
           elsif CLK='1' and CLK'event then
                xi<= RESIZE(signed(X&"000"),17);
                xi_1<= xi;
                xi_2<= xi_1;
                xi_3<= xi_2;
                mb  <= b*(xi + qi_1);
                qi_1<= xi_1 - mb(27 downto 11);
                qi_2<= qi_1;
                mbp <= qi_2 + mb(27 downto 11);

                yt:= mbp + xi_3;
                if yt(16 downto 13) > max  then
                     yi<= x"7ff";
                elsif yt(16 downto 13) < min  then
                     yi<= x"800";
                else
                     yi<=yt(14 downto 3);
                end if;
           end if;
      end process;
       Y<= std_logic_vector(yi);

end synt;
```

The test results of the filter model in the form of the magnitude frequency response, logarithmic frequency response and phase-frequency response are shown in Fig. 13. In this case, the input signal has an amplitude of 2000 $<2^{11}$, the sampling frequency is 1000 arbitrary units.

The charts show that the filter has a suppression level of 21 dB, is really half-band filter (at a frequency of 250 = 1000/4, the transfer function is $1459/2000 \approx \sqrt{0{,}5}$), its phase characteristic approximately linear, but at a frequency of 332 the phase is changed sharply by an angle π, and at this frequency, just a collapse occurs in the frequency response.

When configuring the filter in the Xilinx Spartan-6 FPGA, the following results were obtained.

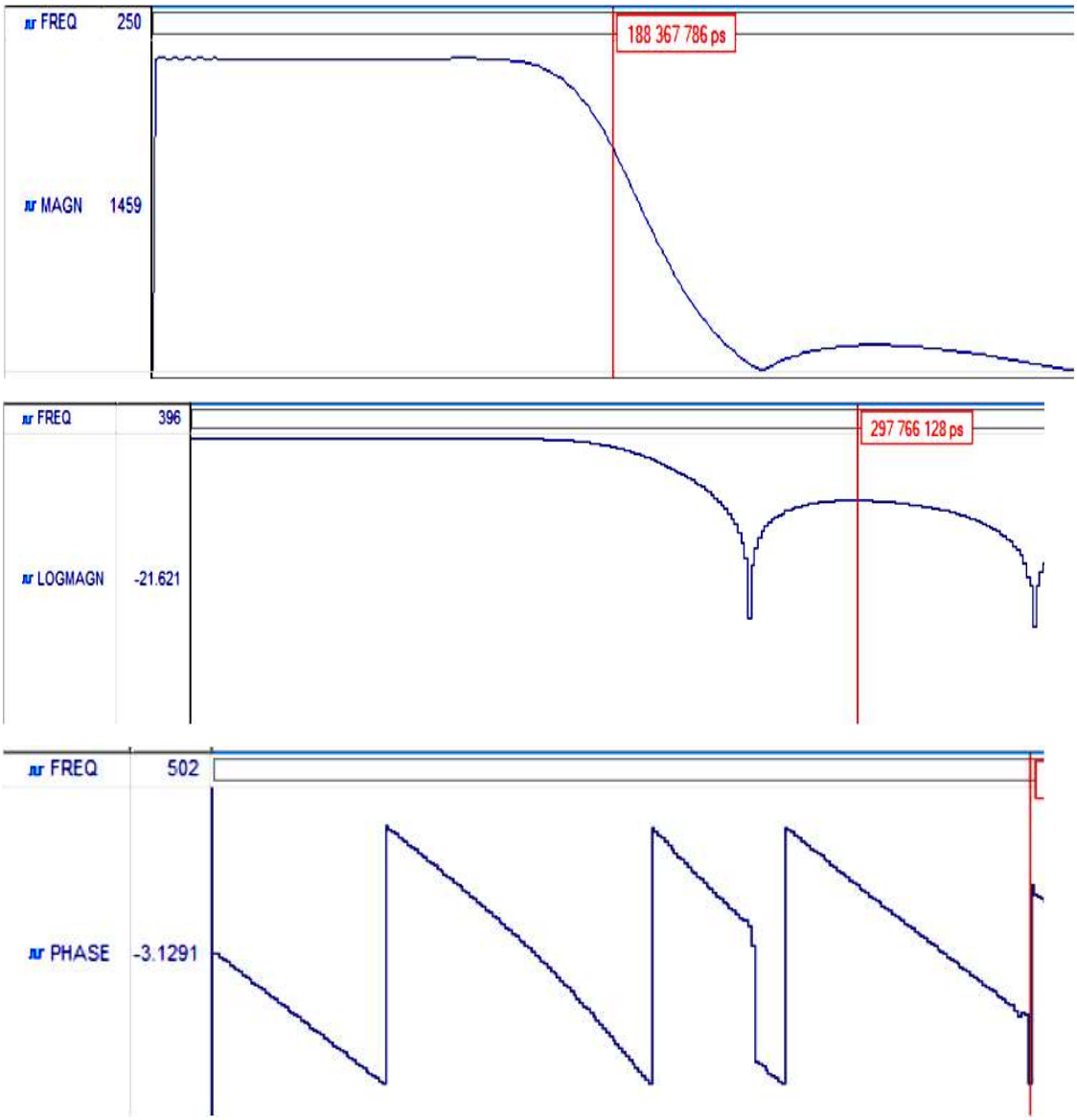


Fig. 13. Magnitude-frequency response, logarithmic frequency response and phase-frequency response of the developed filter

Device Utilization Summary			
Slice Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Registers	125	18,224	1%
Number used as Flip Flops	125		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	91	9,112	1%
Number used as logic	68	9,112	1%
Number using O6 output only	59		
Number using O5 output only	0		
Number using O5 and O6	9		
Number used as ROM	0		
Number used as Memory	0	2,176	0%
Number used exclusively as route-thrus	23		
Number with same-slice register load	23		
Number with same-slice carry load	0		
Number with other load	0		
Number of occupied Slices	31	2,278	1%
Nummber of MUXCYs used	72	4,556	1%
Number of LUT Flip Flop pairs used	107		
Number of DSP48A1s	1	32	3%

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
ts_clk = PERIOD TIMEGRP "CLK" 7.93 ns HIGH 50%	SETUP	0.000ns	7.930ns	0	0
	HOLD	0.382ns		0	0

So, the hardware costs of the synthesized filter are 31 CLB slices and 1 block DSP48. The filter clock frequency reaches f_C = 1 / 7.93 = 126 MHz.

Laboratory Excercise 2

Designing a digital filter without multiplication blocks

Goal: To gain knowledge and skills in the development and testing of high-speed digital filters in FPGAs using techniques of the hardware costs reducing.

Theoretical information

Mask filters

When the digital filter cascades are connected in series, the resulting frequency response is the intersection of the frequency responses of these cascades. It is said that the cascade frequency response masks the frequency response of other cascades, that is, such a cascade is a masking filter (Fig. 14). Thanks to masking, the resulting filter, consisting of simple filter cascades, has a high-quality frequency response.

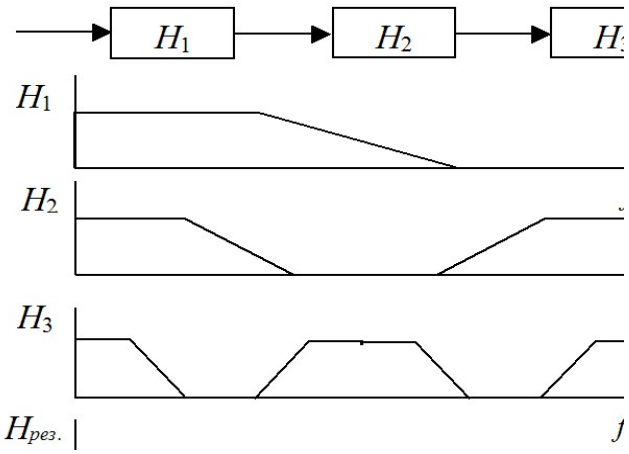


Fig. 14. An example of a three cascade filter with the mask filters

Multiple delay filters

Each term z^{-k} in the transfer function H (z) in the filter signal graph corresponds to a delay of k cycles or a chain of k delay registers in the filter structure. If the number of delay registers in the filter is increased in n times, a

filter with frequency response $H_n(z) = H(zn)$ is obtained. The frequency response of this filter has the same form as the prototype filter $H(z)$, but in the range of 0 — f_S it is repeated n times, where f_S is the sampling frequency. For example, in fig. 14 $H_1 = H(z)$, $H_2 = H(2z)$, $H_3 = H(4z)$.

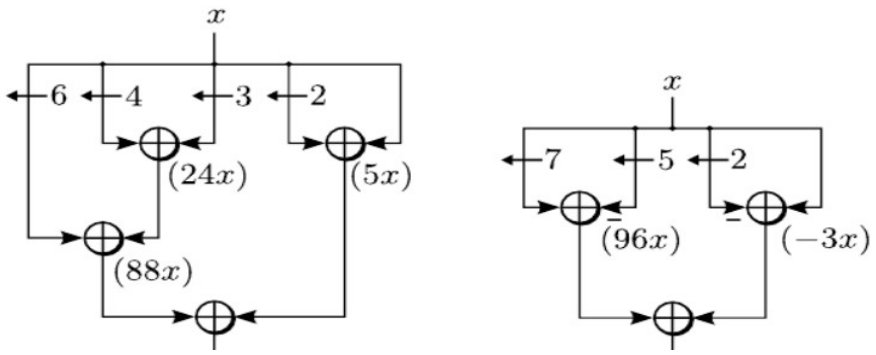
Replacing a multiplication block with a constant multiplier

Most multiplication blocks in DSP are the blocks that multiply to the constants. If a constant can be represented in a canonical number system with a small number of nonzero digits, then the general purpose multiplier is worth to be replaced by an application-specific multiplier in the form of a tree of partial product adders.

For example, consider a constant $y = 93_{10} = 101110_{12}$. Then the product is

$$y \cdot x = 93x = (2^6 + 2^4 + 2^3 + 2^2 + 1)x,$$

that is, when representing the factor y in the binary form, we have 5 nonzero digits. The multiplication by such a factor consists of shifting the datum x by the corresponding number of bits and adding them on the tree of 4 adders. A diagram of such a tree is shown in Fig. 15a, on which the horizontal arrows show a shift to the right by the corresponding number of bits.



a) b) c)

Fig. 15. Examples of the constant multipliers

If the constant is represented in the signed canonical notation, the number of nonzero digits decreases. It should be noted that the complexity of the adder and the subtractor are the same. For instance,

$y = 93_{10} = 1011101_2 = 1100\bar{1}01_2 = 10\bar{1}00\bar{1}01_2 = (2^7 - 2^5 - 2^2 + 1)$.

Then the number of adders decreases to three, as in Fig. 15, b. You can still improve the scheme after factorizing the coefficient. For instance,

$10\bar{1}00\bar{1}01_2 = (2^1 + 1)\cdot(2^5 - 1)$.

Then the number of adders decreases to two, as in Fig. 15, c.

Task for work

To develop a VHDL project of a digital filter without the multiplication blocks with the following transfer function

$H(z) = H_4(z)\cdot H_M(z)$,

where $H_4(z) = H_3(zk)$,

$H_3(z) = (H_1(z) + H_2(z))/2$,

$H_1(z)$ is the same as in the laboratory work 1 and is implemented in the corresponding SDF, and $H_M(z)$, $H_2(z)$, *a, b, k* are selected from Table 3.

The filter model is a description of a given SDF in VHDL. SDF is necessarily optimized by the retiming and pipelining methods, as this is facilitated by the use of multiple delays in the SDF feedbacks.

The testing of the developed filter is performed on a test bench, such as in fig. 8.

Also, the filter has to be synthesized with the placement and routing in selected FPGA CAD (Xilinx or Intel) for FPGA, which is chosen arbitrarily.

The protocol of the laboratory work in which the following items must be:

– filter algorithm, and optimized SDF;

– VHDL description of the filter, which should have appropriate comments indicating the author and explanations of the algorithm execution.

– frequency response charts in the linear and logarithmic scales derived from the filter testing;

Table 3. Parameters and functions for the laboratory work 2

Variant №	***a***	***b***	$H_2(z)$	***k***	$H_M(z)$
1	–0,3125	0,75	–1	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/8$
2	–0,125	0,75	–1	2	$(1-z^{-1}+z^{-2}-z^{-3}+z^{-4})/8$
3	–0,625	0,75	–1	2	$(1+4z^{-1}+6z^{-2}+4z^{-3}+z^{-4})/16$
4	–0,875	0,75	–1	2	$(1-4z^{-1}+6z^{-2}-4z^{-3}+z^{-4})/16$
5	–0,3125	0,5	$-z^{-1}$	2	$(1+4z^{-1}+6z^{-2}+4z^{-3}+z^{-4})/16$
6	–0,75	0,5	$-z^{-1}$	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/8$
7	–0,5	0,5	$-z^{-1}$	2	$(-1+3z^{-1}+5z^{-2}+3z^{-3}-z^{-4})/8$
8	–0,25	0,5	$-z^{-1}$	2	$(1+5z^{-1}+10z^{-2}+10z^{-3}+5z^{-4}+z^{-5})/32$
9	–0.875	0,5	z^{-1}	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6}+z^{-7})/8$
10	0.5	0,5	z^{-1}	2	$(1+z^{-1})(1+z^{-1}+z^{-2}+z^{-3})/8$
11	–0,25	0,25	z^{-1}	3	$(2+5z^{-1}+7z^{-2}+5z^{-3}+2z^{-4})/32$
12	–0,5	0,25	z^{-1}	3	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5})/8$
13	–0,125	0,25	z^{-1}	3	$(1+z^{-1}+2z^{-2}+z^{-3}+z^{-4})/8$
14	–0,75	0,25	z^{-1}	3	$(1-z^{-1}+z^{-3}-z^{-4})/4$
15	–0,75	0,25	z^{-1}	3	$(1+0.7z^{-1}-0.7z^{-3}-z^{-4})/4$
16	–0,625	0,25	z^{-1}	3	$(1-z^{-1}+z^{-2}-z^{-3}+z^{-4}-z^{-5})/8$
17	–0,75	0,25	z^{-1}	3	$(1-0.7z^{-1}+0.7z^{-3}-z^{-4})/4$
18	–0,5	0,25	z^{-1}	2	$(1+z^{-1}+z^{-3}+z^{-4})/4$
19	0.25	0,25	z^{-1}	2	$(-1+z^{-1}+z^{-3}-z^{-4})/4$
20	–0,625	0,25	z^{-1}	2	$(1-z^{-1}+z^{-3}-z^{-4})/4$
21	–0,5	0,25	z^{-1}	2	$(-1+2z^{-2}-z^{-4})/4$
22	–0,75	0,25	z^{-1}	2	$(1+2z^{-1}+2z^{-2}+2z^{-3}+z^{-4})/8$
23	–0,6	0,25	z^{-1}	2	$(1-2z^{-1}+2z^{-2}-2z^{-3}+z^{-4})/8$
24	0	0,25	z^{-1}	2	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4}+z^{-5}+z^{-6})/8$
25	–0.1	0,25	z^{-1}	2	$(1-z^{-2}+z^{-4}-z^{-6})/4$
26	–0.15	0,25	z^{-1}	3	$(1+1.4z^{-1}+z^{-2}-z^{-4}-1.4z^{-5}-z^{-6})/8$
27	–0,5	0,25	z^{-1}	3	$(1+z^{-1}+z^{-2}+z^{-3}+z^{-4})/8$
28	–0,625	0,25	z^{-1}	3	$(-3-2z^{-1}+5z^{-2}+5z^{-3}-2z^{-4}-3z^{-5})/32$
29	–0,5	0,25	z^{-1}	3	$(1+5z^{-1}+10z^{-2}+10z^{-3}+5z^{-4}+z^{-5})/32$
30	–0,75	0,25	z^{-1}	3	$(3-2z^{-1}+5z^{-2}+5z^{-3}-2z^{-4}+3z^{-5})/32$

– synthesis results in the form of a screenshot showing the hardware costs and the minimum period of the clock frequency for the selected FPGA.

The variant number for this lab is the same as the student number in the group list.

The bit depth of input, output and intermediate data is the same as in the laboratory work 1.

Execution example

Consider SDF of the IIR filter part such as in Fig. 11 and it is necessary to develop LPF with $a = 0$, $b = 0.5625$, $k = 2$; $H_M(z) = (1 + 4z^{-1} + 6z^{-2} + 4z^{-3} + z^{-4})/16$.

The input data bit width is 12.

The filter SDF is shown in fig. 16. It consists of the IIR part (left) and the FIR part (right).

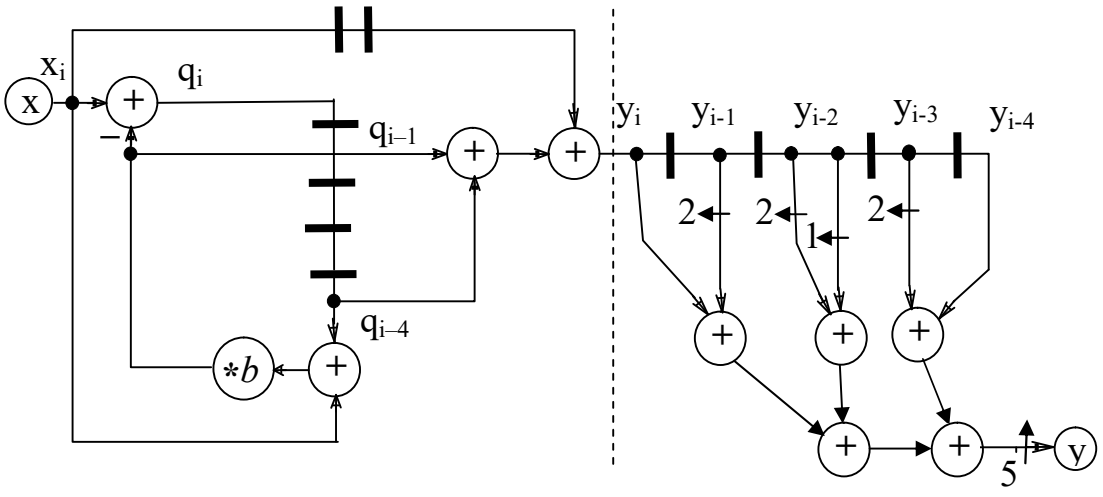


Fig. 16. Low-pass filter SDF

This SDF is then retimed using the pipelining technique. Also, the multiplication by b is replaced by an application-specific adder-based multiplier. Then, $b = 0{,}4375 = 0{,}100\bar{1}$ means that the multiplication is performed as the subtraction of the operand shifted by 4 bits from the operand shifted by 1 bit. Similarly are implemented multiplication by coefficients in the FIR part.

The resulting filter SDF is shown in Fig. 17. It shows all the signals that are involved in the calculations. The filtering result is divided by 32 using a shift right to 5 bits, taking into account the transmission coefficients of the IIR and FIR parts. The critical path is minimized to the delay of one adder due to the retiming and pipelining, as well as the replacement of multiplication by addition.

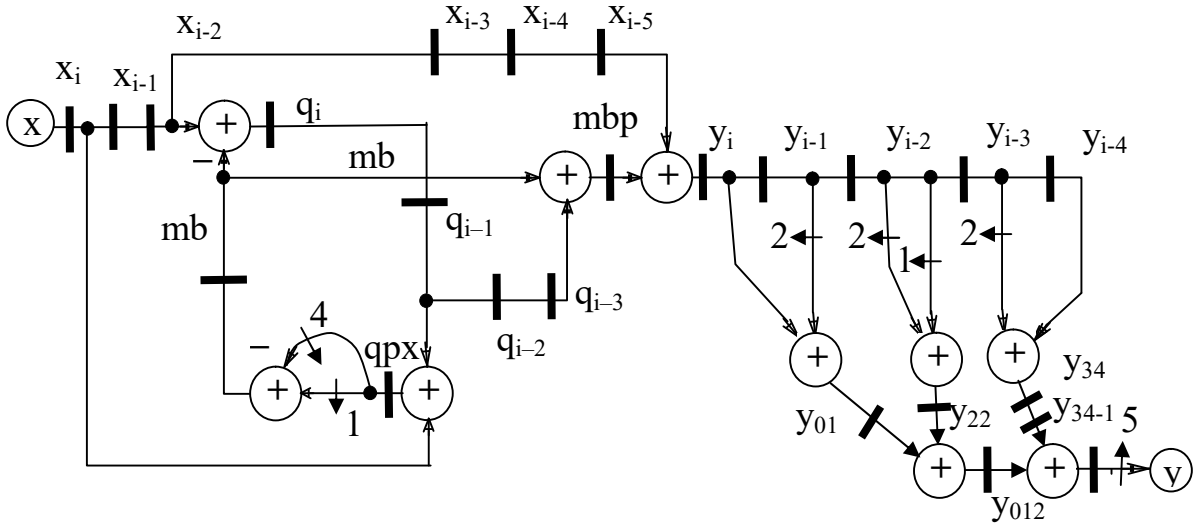


Fig. 17. Low-pass filter pipelined SDF

The bit width of the intermediate data in the FIR part is selected: $n_c = n_д + n_x + 3 = 2 + 12 + 3 = 17$. Taking into account the formula (12), the intermediate data bit width of the FIR part is $n_y = n_c + 4 = 17 + 4 = 21$. The resulting VHDL filter description is presented below.

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.NUMERIC_STD.all;
entity LPF_HB_LAB2 is
     port(
          CLK : in STD_LOGIC;
          RST : in STD_LOGIC;
          X : in STD_LOGIC_VECTOR(11 downto 0);
          Y : out STD_LOGIC_VECTOR(11 downto 0)
          );
end LPF_HB_LAB2;

architecture synt of LPF_HB_LAB2 is
     constant max:signed(3 downto 0):="0111";
     constant min:signed(3 downto 0):="1100";
     signal xi,xi_1,xi_2,xi_3,xi_4,xi_5:signed(16 downto 0);
     signal qi,qi_1,qi_2,qi_3,mb,qpx,mbp:signed(16 downto 0);
     signal yi,yi_1,yi_2,yi_3,yi_4:signed(16 downto 0);
     signal y01,y22,y34,y012,y34_1:signed(20 downto 0);
     signal ys:signed(11 downto 0);
begin
     IIR:process(CLK,RST)
     begin
          if RST = '1' then
```

```
            xi<=(others=>'0');       xi_1<=(others=>'0');
            xi_2<=(others=>'0');     xi_3<=(others=>'0');
            xi_4<=(others=>'0');     xi_5<=(others=>'0');
            yi<=(others=>'0');       qi<=(others=>'0');
            qi_1<=(others=>'0');     qi_2<=(others=>'0');
            qi_3<=(others=>'0');     qpx<=(others=>'0');
            mb<=(others=>'0');       mbp<=(others=>'0');
        elsif CLK='1' and CLK'event then
            xi<= RESIZE(signed(X&"000"),17);
            xi_1<= xi;
            xi_2<= xi_1;
            xi_3<= xi_2;
            xi_4<= xi_3;
            xi_5<= xi_4;
            qpx <= xi + qi_1;
            mb  <= shift_right(qpx,1) - shift_right(qpx,4);
            qi  <= xi_2 - mb;
            qi_1<= qi;
            qi_2<= qi_1;
            qi_3<= qi_2;
            mbp <= qi_3 + mb;
            yi<= mbp + xi_5;
        end if;
    end process;

    FIR:process(CLK,RST)
      variable yt:signed(20 downto 0);
    begin
        if RST = '1' then
            yi_1<=(others=>'0');     yi_2<=(others=>'0');
            yi_3<=(others=>'0');     yi_4<=(others=>'0');
            y01 <=(others=>'0');     y012 <=(others=>'0');
            y22 <=(others=>'0');     y34 <=(others=>'0');
            y34_1<=(others=>'0');    ys   <=(others=>'0');
        elsif CLK='1' and CLK'event then
            yi_1<= yi;
            yi_2<= yi_1;
            yi_3<= yi_2;
            yi_4<= yi_3;
            y01 <= yi + resize((yi_1 &"00"),21) ;
            y22 <= resize((yi_2 &"00"),21) + (yi_2 & "0");
            y34 <= yi_4 + resize((yi_3 & "00"),21) ;
            y34_1<= y34;
            y012 <= y01 + y22;
            yt:= y012 + y34_1;
            ys<=yt(19 downto 8);
        end if;
    end process;
          Y<= std_logic_vector(ys);
end synt;
```

The test results derived on the test bench are shown in Fig.18.

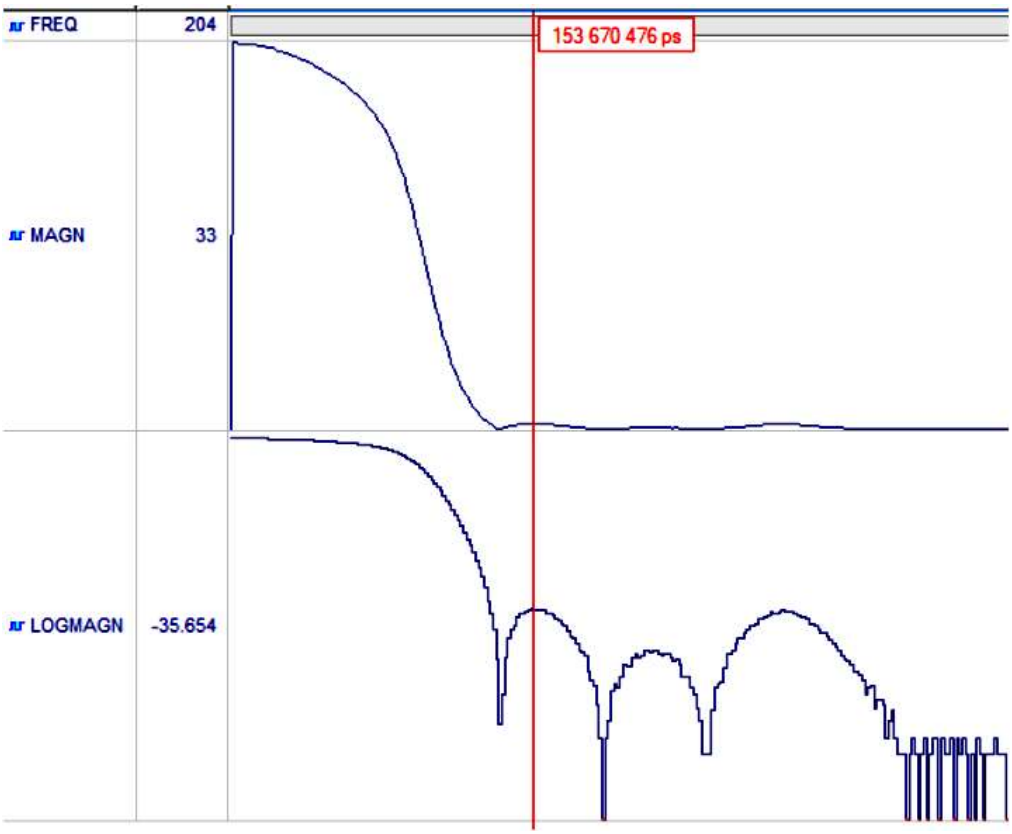


Fig. 18. The magnitude-frequency response and the logarithmic frequency response of the filter in Fig. 17

From the obtained charts, one can see that the filter has a suppression level of 35.6 dB, a cutoff frequency of 0.112, and a start frequency of the suppression band of 0.175.

When configuring the filter in the Xilinx Spartan-6 FPGA, the following results are derived.

Slice Logic Utilization	Used	Available	Utilization
Number of Slice Registers	382	18,224	2%
Number used as Flip Flops	382		
Number used as Latches	0		
Number used as Latch-thrus	0		
Number used as AND/OR logics	0		
Number of Slice LUTs	263	9,112	2%
Number used as logic	173	9,112	1%
Number using O6 output only	169		
Number using O5 output only	0		
Number using O5 and O6	4		
Number used as ROM	0		
Number used as Memory	1	2,176	1%
Number used as Dual Port RAM	0		
Number used as Single Port RAM	0		
Number used as Shift Register	1		
Number using O6 output only	1		
Number using O5 output only	0		
Number using O5 and O6	0		
Number used exclusively as route-thrus	89		
Number with same-slice register load	88		
Number with same-slice carry load	1		
Number with other load	0		
Number of occupied Slices	77	2,278	3%
Nummber of MUXCYs used	192	4,556	4%
Number of LUT Flip Flop pairs used	286		
Number with an unused Flip Flop	9	286	3%
Number with an unused LUT	23	286	8%
Number of fully used LUT-FF pairs	254	286	88%

Constraint	Check	Worst Case Slack	Best Case Achievable	Timing Errors	Timing Score
ts_clk = PERIOD TIMEGRP "CLK" 2.8 ns HIGH 50%	SETUP	0.248ns	2.552ns	0	0
	HOLD	0.388ns		0	0
	MINPERIOD	0.134ns	2.666ns	0	0

So, the hardware volume of the synthesized filter is 77 CLB slices, including 382 triggers and 263 LUTs.

The filter maximum clock frequency reaches f_C = 1/2.666 = 375 MHz. This is almost three times higher than for a filter that uses a multiplication block (see Laboratory exercise 1). Thus, the technique of using application-specific multiplication blocks not only reduces the hardware costs (DSP48 multiplication blocks, each of which is equivalent to 20 adders) but also significantly increases the filter performance.

Recommended literature

1. Сергиенко А.М. VHDL для проектирования вычислительных устройств. Киев: ЧП "Корнейчук", ТИД ДС, 2003. — 208 с.

2. Сергієнко А.М. Генератор рекурсивних фільтрів без блоків множення. 2014. [електронний ресурс] http://kanyevsky.kpi.ua/GEN_MODUL/APgen/APMF_help_ukr.php

3. Сергієнко А. М., Сергієнко А.А. Методика проектування цифрових фільтрів з застосуванням VHDL. //Праці 3 міжнародної конференції InfoCom'2016, 1 – 2 грудня 2016 р. -К.:НТУУ "КПІ", ВПІ "Політехніка". – 2016. –С. 56-57. [електронний ресурс] https://iconfs.net/w.infocom2016/metodyka-proektuvannya-tsyfrovykh-filtriv-z-zastosuvannyam-vhdl

4. Сергієнко А.М., Виноградов Ю.М., Лесик Т.М. Цифрова обробка сигналів. Комп'ютерний практикум мовою VHDL. – Київ. – 2012. – 106 с. [електронний ресурс] http://kanyevsky.kpi.ua/wp-content/uploads/2017/11/DSP_LabS.pdf

5. Schlichthärle D. Digital Filters: Basics and Design. – Springer. Berlin Heidelberg, –2011. – 527 p.

6. Khan S. A. Digital Design of Signal Processing Systems. A Practical Approach. – Wiley. – 211. – 586 p. – Available at http://dspace.bhos.edu.az/jspui/bitstream/123456789/1146/1/%5BShoab_Ahmed_Khan%5D_Digital_Design_of_Signal_Process.pdf