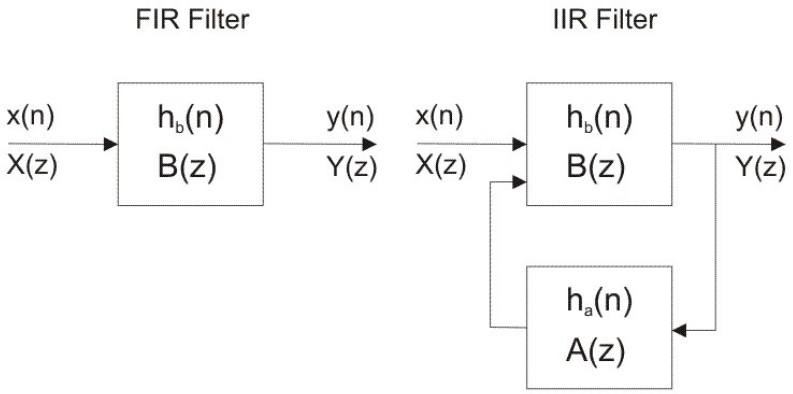
3.1. IIR filtrai

IIR filtrai yra skaitmeniniai filtrai, turintys neribotą impulsų atsaką. Skirtingai nuo FIR filtrų, jie turi grįžtamąjį ryšį (rekursinė filtro dalis), todėl yra žinomi kaip rekursiniai skaitmeniniai filtrai.



1 pav. FIR ir IIR filtrų struktūrinė schema

Dėl šios priežasties IIR filtrai turi daug geresnį dažninį atsaką nei tos pačios eilės FIR filtrai. Skirtingai nuo FIR filtrų, jų fazinė charakteristika nėra tiesinė, o tai gali sukelti problemų sistemose, kurioms reikalingas fazinis tiesiškumas. Todėl nėra pageidautina naudoti IIR filtrus skaitmeniniam signalų apdorojimui, kuomet fazė yra esminė.

Priešingu atveju, kai tiesinės fazės charakteristikos nėra svarbios, IIR filtrų naudojimas yra tinkamas sprendimas.

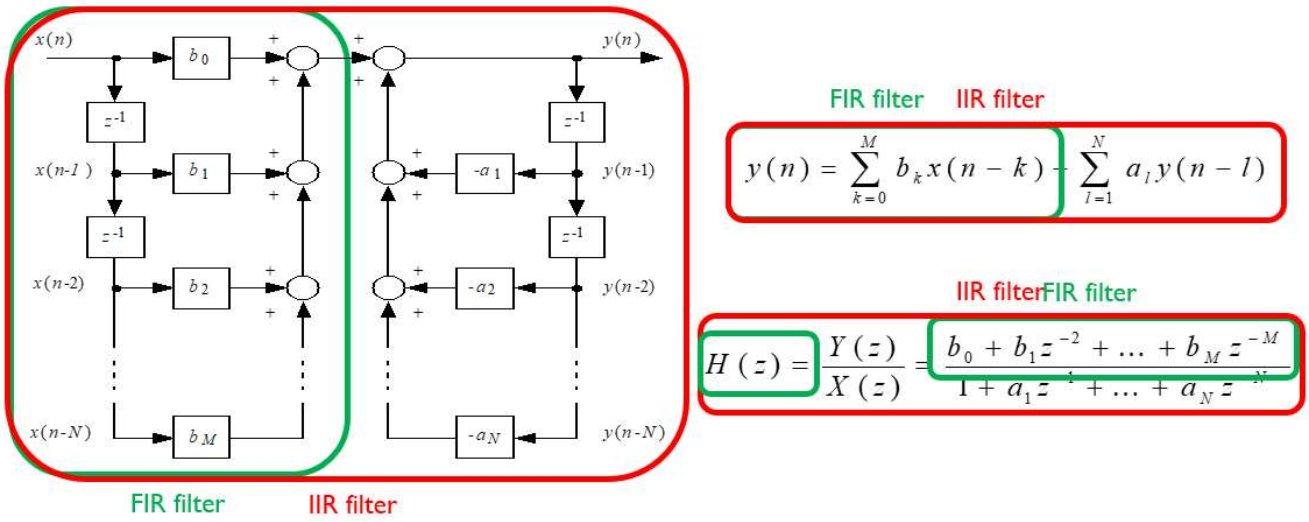
Kita svarbi savybė - tai galimas nestabilumas, būdingas tik IIR filtrams. FIR filtrai neturi tokios problemos, nes jie neturi grįžtamojo ryšio dalies. Dėl šios priežasties visuomet po projektavimo būtina patikrinti, ar gautas IIR filtras yra stabilus, ar ne.

IIR (Infinite Impulse Response) skaitmeninius filtrus galima apibūdinti pagal įvesties/išvesties skirtumo lygtį:

$$y(k) = \sum_{i=0}^{N} b_i x(k-i) - \sum_{i=1}^{N} a_i y(k-i) \quad (1)$$

- Konstantos $b_0, b_1, \ldots, b_N$ ir $a_1, \ldots, a_N$ – filtro koeficientai.
- N – filtro eilė. Iš viso yra $2N+1$ filtro koeficientų.
- IIR filtro projektavimas apima filtro eilės nustatymą bei filtro koeficientų, kurie atitinka nustatytus reikalavimas, suradimą.
- Filtro išėjimo signalo suradimas – MAC (*multiply and accumulate*) operacija: esama įėjimo signalo vertė ir N buvusių įėjimo signalo bei išėjimo signalo verčių sudauginama su filtro koeficientais. Gautas daugybos rezultatas sudedamas tarpusavyje.
- Norint įgyvendinti filtrą, filtro koeficientai, esama signalo vertė, tam tikras baigtinis ankstesnių įėjimo signalo verčių bei atitinkamas ankstesnių išėjimo signalo verčių turi būti išsaugoti atmintyje.
- IIR filtras yra rekursinis; tai reiškia, kad filtro išvesties signalas priklauso nuo filtro įvesties signalo ir ankstesnių filtro išvesties signalo verčių. Tuo tarpu FIR filtro išvesties signalas priklauso tik nuo įvesties signalo, todėl jis nėra rekursinis.

IIR filtro struktūrinė schema, perdavimo lygtis bei z-perdavimo funkcija pateikta 1 pav. Analizuojant struktūrinę schema, galima išskirti dalį, kuri atitinka nerekursinį FIR filtrą.



2 pav. IIR filtras ir jo perdavimo funkcija

IIR filtro vienetinio impulso perdavimo funkcija

FIR filtro impulsinis atsakas yra tiesiog lygus filtro koeficientams, todėl yra baigtinis laike. Tačiau tai netinka IIR filtro atveju. IIR filtro impulso atsakas yra begalinis laike; jei filtras yra stabilus, $h(k) \rightarrow 0$ jei $k \rightarrow \infty$.

IIR filtro perdavimo funkcija

IIR filtro perdavimo funkciją galima gauti remiantis įėjimo/išėjimo skirtumo lygtimi, pasinaudojus z-transformacijos laiko poslinkio savybe.

IIR filtro stabilumas

Diskretinio laiko sistemos yra stabilios, jei kiekvieno poliaus dydis yra mažesnis už vienetą. IIR filtro poliai yra N vardiklio šaknys. Skirtingai nei FIR filtrų atveju, IIR filtrai ne visuomet bus stabilūs.

IIR filtro išėjimo signalas

IIR filtro išėjimo signalas gali būti apskaičiuotas skirtingais metodais, pavyzdžiui:

1. Rekursyviai, naudojant įvesties/išvesties skirtumo lygtį
2. Kompozicijos pagalba, naudojant išraišką $y(k) = x(k)*h(k)$
3. Z-transformacijos inversija Y(z) = H(z)X(z)
4. Programinių paketų pagalba, pavyzdžiui, Matlab
5. Naudojant Greitąją Furjė transformaciją (FFT)

IIR Filtrų projektavimas

Galima išskirti du pagrindinius IIR filtrų projektavimo būdus: netiesioginis ir tiesioginis.

Netiesioginis IIR filtro projektavimas

Netiesioginio projektavimo metodas paremtas pirmiausia atitinkamo analoginio filtro suprojektavimu. Vėliau analoginis filtras, atitinkantis iškeltus reikalavimas, paverčiamas į skaitmeninį filtrą su atitinkančiomis charakteristikos. Veiksmų seka:

1. Skaitmeninio filtro specifikacijas paversti analoginėmis specifikacijomis.
2. Paversti analogines specifikacijas į žemo dažnio prototipo specifikacijas.
3. Suprojektuoti žemo dažnio analoginio filtro prototipą (Batervorto, Čebyševo I ar II, Elipsinis).
4. Paversti žemo dažnio filtro prototipą į reikiamą analoginį filtrą.
5. Paversti analoginį filtrą į atitinkamą skaitmeninį filtrą.

Ankstyvosiose skaitmeninių filtrų projektavimo stadijose buvo gausu prieinamos informacijos apie analoginių filtrų projektavimą: žemo dažnio Batervorto, Čebyševo, Elipsinių filtrų dažninį atsaką aprašančios lygtys, bei lygtys, skirtos transformacijai tarp skirtingų tipų filtrų – iš žemo dažnio į aukšto dažnio filtrus ir kt.). Taigi, savaime suprantama buvo prasmės projektuoti gerą analoginį filtrą, atitinkantį reikiamas specifikacijas ir tuomet šį filtrą paversti į atitinkamą skaitmeninį filtrą.

Vėliau išpopuliarėjo iteraciniai kompiuteriais grįsti skaičiavimo metodai, skirti surasti reikiamus IIR filtro koeficientus. Įvairūs programiniai paketai automatizuoja „tradicinius“ projektavimo metus ir leidžia vartotojui pasirinkti norimą variantą iš įvairių analoginių filtrų topologijų, pavyzdžiui, Batervorto, Čebyševo, Beselio ir kt.

Tiesioginis IIR filtro projektavimas

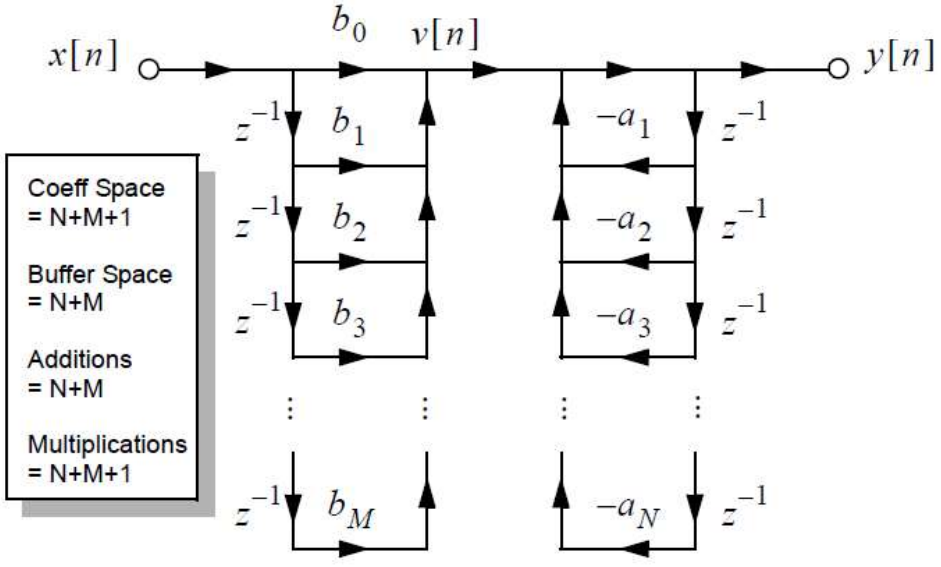
Tiesioginis projektavimo metodas apima skaitmeninio filtro projektavimą tiesiogiai (be tarpinio analoginio filtro projektavimo etapo) taip, kad jis atitiktų iškeltus reikalavimus.

IIR filtrų struktūros

IIR filtrai gali būti įgyvendinti remiantis įvairiomis topologijomis. Dažniausiai sutinkamos yra: tiesioginė I forma, tiesioginė II forma, kaskadinė, lygiagrečioji.

Tiesioginė I forma (Direct Form I)

Filtras pagal 1 formulę gali būti realizuotas naudojant tiesioginę I formą.



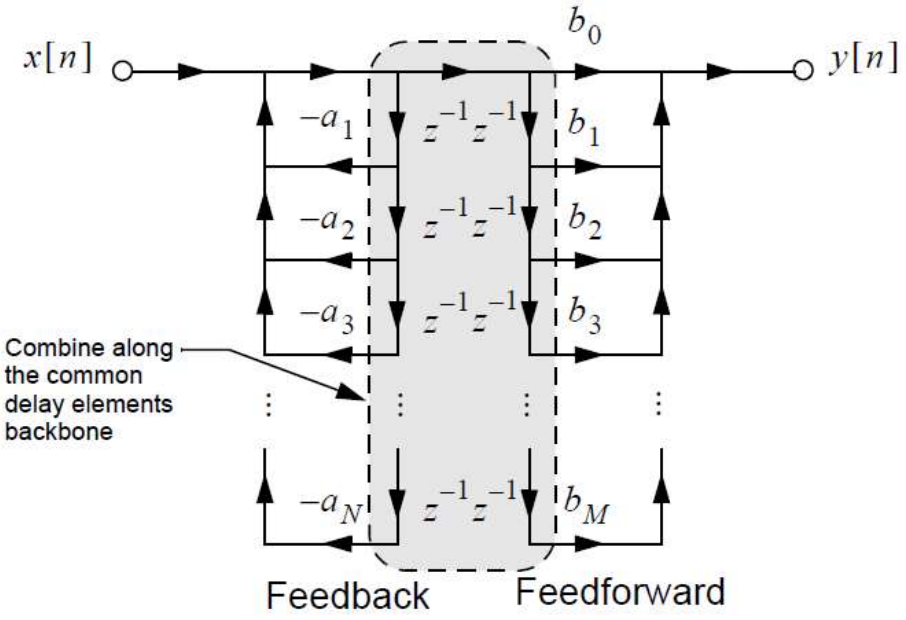
3 pav. IIR filtro Tiesioginė I forma

Esant *N-os* eilės filtrui, struktūra turi iš viso *2N* vėlinimo elementų (pažymėtų z^{-1}). Jei, pavyzdžiui, filtras bus 2-eilės, pateiktoje struktūroje bus 4 vėlinimo elementai.

Tiesioginė II forma (Direct form II)

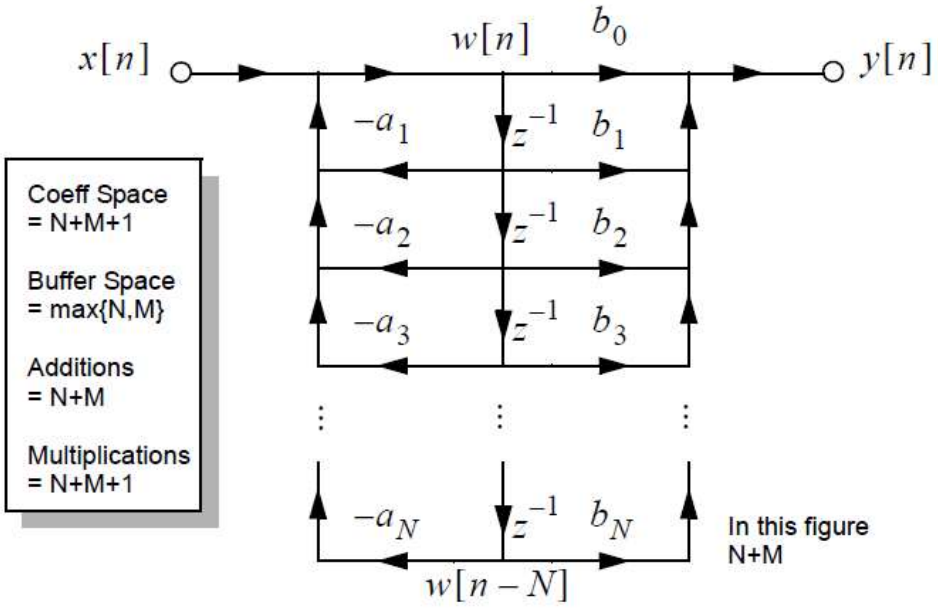
Tiesioginė II forma yra viena dažniausiai naudojamų struktūrų realizuojant IIR filtrus. Ši struktūra reikalauja perpus mažiau vėlinimo elementų todėl yra efektyvesnė už 1-ą formą.

Sukeitus 1-oje formoje rekursinę filtro dalį su tiesiogine, gausime struktūrą:



4 pav. Transformuota tiesioginė filtro struktūra

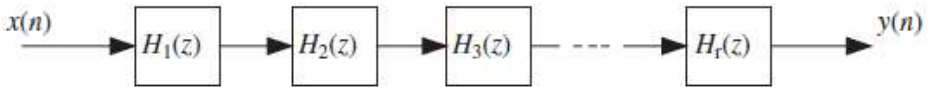
Iš transformuotos filtro struktūros matome, kad šiuo atveju galima panaudoti bendrą vėlinimo liniją. Galutinė tiesioginė II forma parodyta 5 pav.



5 pav. IIR filtro Tiesioginė II forma

Kaskadinė struktūra

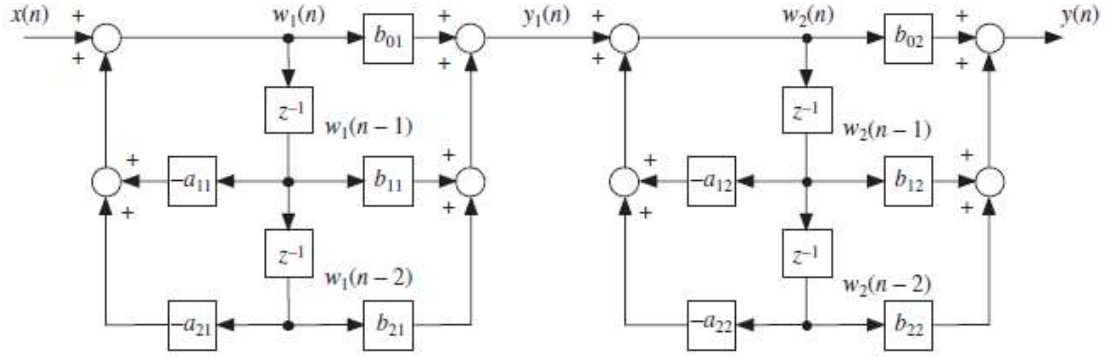
Sistemos perdavimo funkcija H(z) – daugianarių polinomų santykis. Šiuos skaitiklio ir vardiklio polinomus galima išskaidyti įvairiais būdais. Vienas populiariausių kelių – naudoti 2-os eilės polinomus.



6 pav. IIR filtro kaskadinė struktūra

6 pav. parodyta kaskadinė IIR filtro struktūra. Šiuo atveju visos sistemos perdavimo funkcija, naudojant 2-os eilės tiesioginės II formos struktūras, bus lygi:

$$H(z) = \prod_{i=1}^{N/2} \frac{b_{0i} + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}}, \tag{2}$$



7 pav. 4-os eilės IIR filtras, sudarytas iš dviejų kaskadinių elementų

Šios struktūros privalumai:

- Išvengiama sudėtingų aukštos eilės polinomo skaičiavimų
- Skaičiavimai atliekami greičiau dėl konvejerio principo – viename bloke gautas tarpinis rezultatas perduodamas sekančiam blokui (tokiu atveju, jei procesoriuje yra keletas MAC)

Kaskadinės filtrų struktūros su 2-os eilės polinomais populiarios realaus laiko SSA, kadangi jas galima projektuoti MATLAB *FilterDesigner* įrankio pagalba, o realizavimui mikrovaldiklyje parengtos CMSIS-DSP bibliotekos.

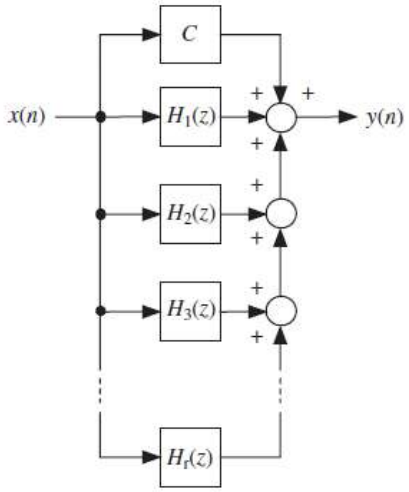
Lygiagrečioji forma

Sistemos perdavimo funkcija gali būti užrašyta:

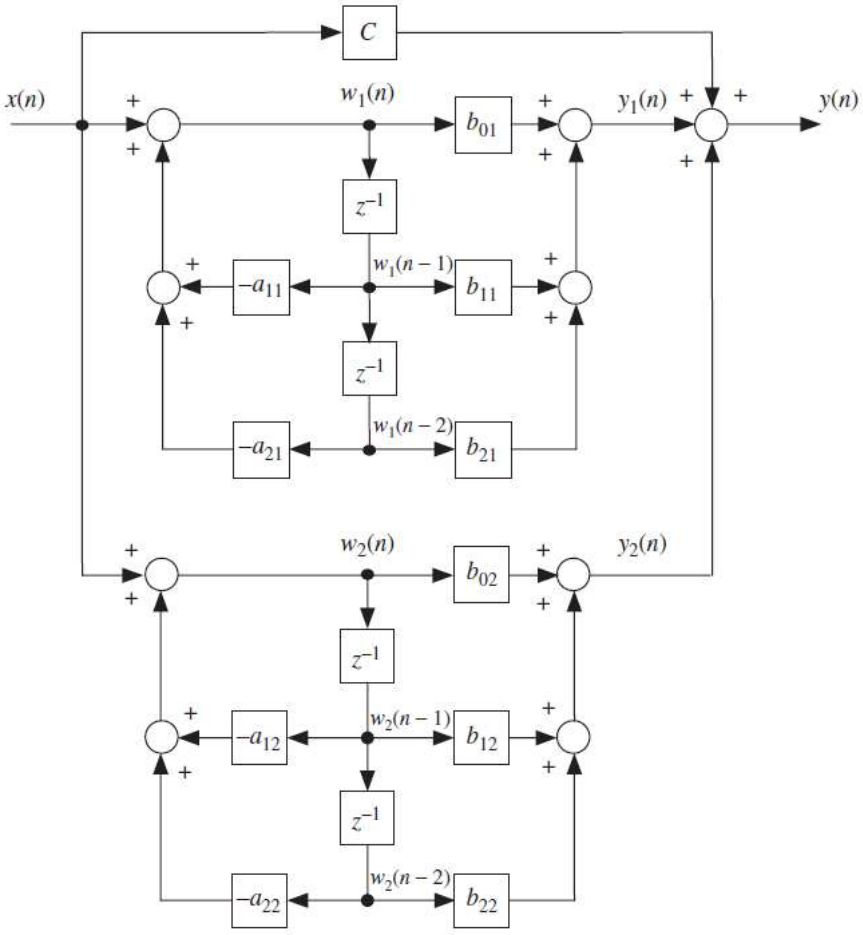
$$H(z) = C + H_1(z) + H_2(z) + \cdots + H_r(z), \tag{3}$$

Kaip ir kaskadinės struktūros atveju, lygiagrečioji struktūra gali būti sudaryta iš 2-eilės tiesioginės II formos blokų. Tuomet perdavimo funkciją galime užrašyti:

$$H(z) = C + \sum_{i=1}^{N/2} \frac{b_{0i} + b_{1i}z^{-1} + b_{2i}z^{-2}}{1 + a_{1i}z^{-1} + a_{2i}z^{-2}}. \tag{4}$$



8 pav. IIR filtro lygiagrečioji struktūra



9 pav. 4-os eilės IIR filtras, sudarytas iš dviejų lygiagrečių elementų

CMSIS-DSP IIR filtrų bibliotekos

CMSIS DSP parengtos šios IIR filtrų bibliotekos:

- *High Precision Q31 Biquad Cascade Filter.*
  - Duomenys Q31 formatu, filtro koeficientai 1.31 formatu, tarpiniai kintamieji 1.63 formatu. Dvigubo tikslumo būsenos kintamieji sumažina kvantavimo paklaidas.
- *Biquad Cascade IIR Filters Using Direct Form I Structure.* Tinkama fiksuoto kablelio skaičiams. Bibliotekos funkcijų palaikomi skaičiai:
  - Slankaus kablelio – 16-os, 32-ų skilčių
  - Fiksuoto kablelio – Q15, Q31
- *Biquad Cascade IIR Filters Using a Direct Form II Transposed Structure.*
  - "Transpose" filtro struktūra reikalauja plataus diapazono tarpinių kintamųjų, todėl palaikomi tik slankaus kablelio skaičiai – 16-os, 32-ų bei 64-ų skilčių.
- *Infinite Impulse Response (IIR) Lattice Filters.*
  - Slankaus kabelio – 32-ų skilčių
  - Fiksuoto kablelio – Q15, Q31

Biquad – angl. santrumpa nuo "biquadratic", reiškianti, kad z-domene filtro perdavimo funkcija yra dviejų antros eilės funkcijų santykis.

Biquad Cascade IIR Filters Using Direct Form I Structure

https://www.keil.com/pack/doc/CMSIS/DSP/html/group__BiquadCascadeDF1.html

This set of functions implements arbitrary order recursive (IIR) filters. The filters are implemented as a cascade of second order Biquad sections. The functions support Q15, Q31 and floating-point data types. Fast version of Q15 and Q31 also available.

The functions operate on blocks of input and output data and each call to the function processes blockSize samples through the filter. pSrc points to the array of input data and pDst points to the array of output data. Both arrays contain blockSize values.

Each Biquad stage implements a second order filter using the difference equation:

```
y[n] = b0 * x[n] + b1 * x[n-1] + b2 * x[n-2] + a1 * y[n-1] + a2 * y[n-2]
```

A Direct Form I algorithm is used with 5 coefficients and 4 state variables per stage.

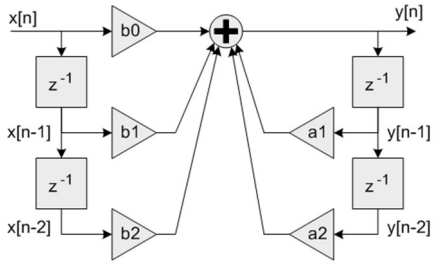


Fig. Single Biquad filter stage

Coefficients *b0, b1* and *b2* multiply the input signal *x[n]* and are referred to as the feedforward coefficients. Coefficients *a1* and *a2* multiply the output signal *y[n]* and are referred to as the feedback coefficients. Pay careful attention to the sign of the feedback coefficients. Some design tools use the difference equation

```
y[n] = b0 * x[n] + b1 * x[n-1] + b2 * x[n-2] - a1 * y[n-1] - a2 * y[n-2]
```

In this case the feedback coefficients *a1* and *a2* must be negated when used with the CMSIS DSP Library.

Higher order filters are realized as a cascade of second order sections. *numStages* refers to the number of second order stages used. For example, an 8th order filter would be realized with *numStages=4* second order stages.

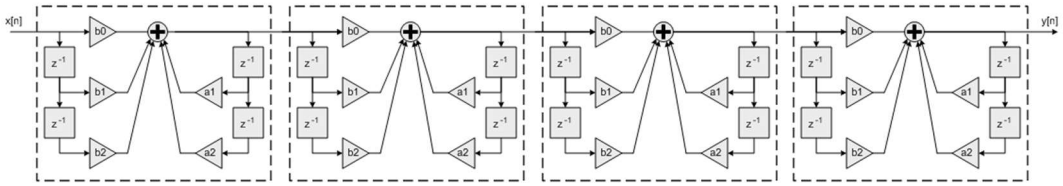


Fig. 8th order filter using a cascade of Biquad stages

A 9th order filter would be realized with *numStages = 5* second order stages with the coefficients for one of the stages configured as a first order filter (*b2 = 0* and *a2 = 0*).

The *pState* points to state variables array. Each Biquad stage has 4 state variables *x[n-1], x[n-2], y[n-1],* and *y[n-2]*. The state variables are arranged in the *pState* array as:

```
{x[n-1], x[n-2], y[n-1], y[n-2]}
```

The 4 state variables for stage 1 are first, then the 4 state variables for stage 2, and so on. The state array has a total length of *4*numStages values*. The state variables are updated after each block of data is processed; the coefficients are untouched.

Instance Structure

The coefficients and state variables for a filter are stored together in an instance data structure.

- A separate instance structure must be defined for each filter.
- Coefficient arrays may be shared among several instances.
- State variable arrays cannot be shared.

There are separate instance structure declarations for each of the 3 supported data types.

Floating-point data type:

```
arm_biquad_casd_df1_inst_f32 S1 = {numStages, pState, pCoeffs};
```

where:

- *numStages* is the number of Biquad stages in the filter;
- *pState* is the address of the state buffer;
- *pCoeffs* is the address of the coefficient buffer.

Init Function

There is also an associated initialization function for each data type. The initialization function performs following operations:

- Sets the values of the internal structure fields.
- Zeros out the values in the state buffer. To do this manually without calling the *init* function, assign the follow subfields of the instance structure: numStages, pCoeffs, pState. Also set all of the values in pState to zero.

Use of the initialization function is optional. However, if the initialization function is used, then the instance structure cannot be placed into a const data section. To place an instance structure into a const data section, the instance structure must be manually initialized. Set the values in the state buffer to zeros before static initialization.

```
void arm_biquad_cascade_df1_init_f32 (  arm_biquad_casd_df1_inst_f32 *  S,
                                        uint8_t                        numStages,
                                        const float32_t *              pCoeffs,
                                        float32_t *                    pState
                                      )
```

Parameters

[in,out]	**S**	points to an instance of the floating-point Biquad cascade structure.
[in]	**numStages**	number of 2nd order stages in the filter.
[in]	**pCoeffs**	points to the filter coefficients.
[in]	**pState**	points to the state buffer.

Coefficient and State Ordering

The coefficients are stored in the array *pCoeffs* in the following order:

```
{b10, b11, b12, a11, a12, b20, b21, b22, a21, a22, ...}
```

where *b1x* and *a1x* are the coefficients for the first stage, *b2x* and *a2x* are the coefficients for the second stage, and so on. The *pCoeffs* array contains a total of *5*numStages* values.

The *pState* is a pointer to state array. Each Biquad stage has 4 state variables *x[n-1], x[n-2], y[n-1],* and *y[n-2]*. The state variables are arranged in the *pState* array as:

```
{x[n-1], x[n-2], y[n-1], y[n-2]}
```

The 4 state variables for stage 1 are first, then the 4 state variables for stage 2, and so on. The state array has a total length of *4*numStages* values. The state variables are updated after each block of data is processed; the coefficients are untouched.

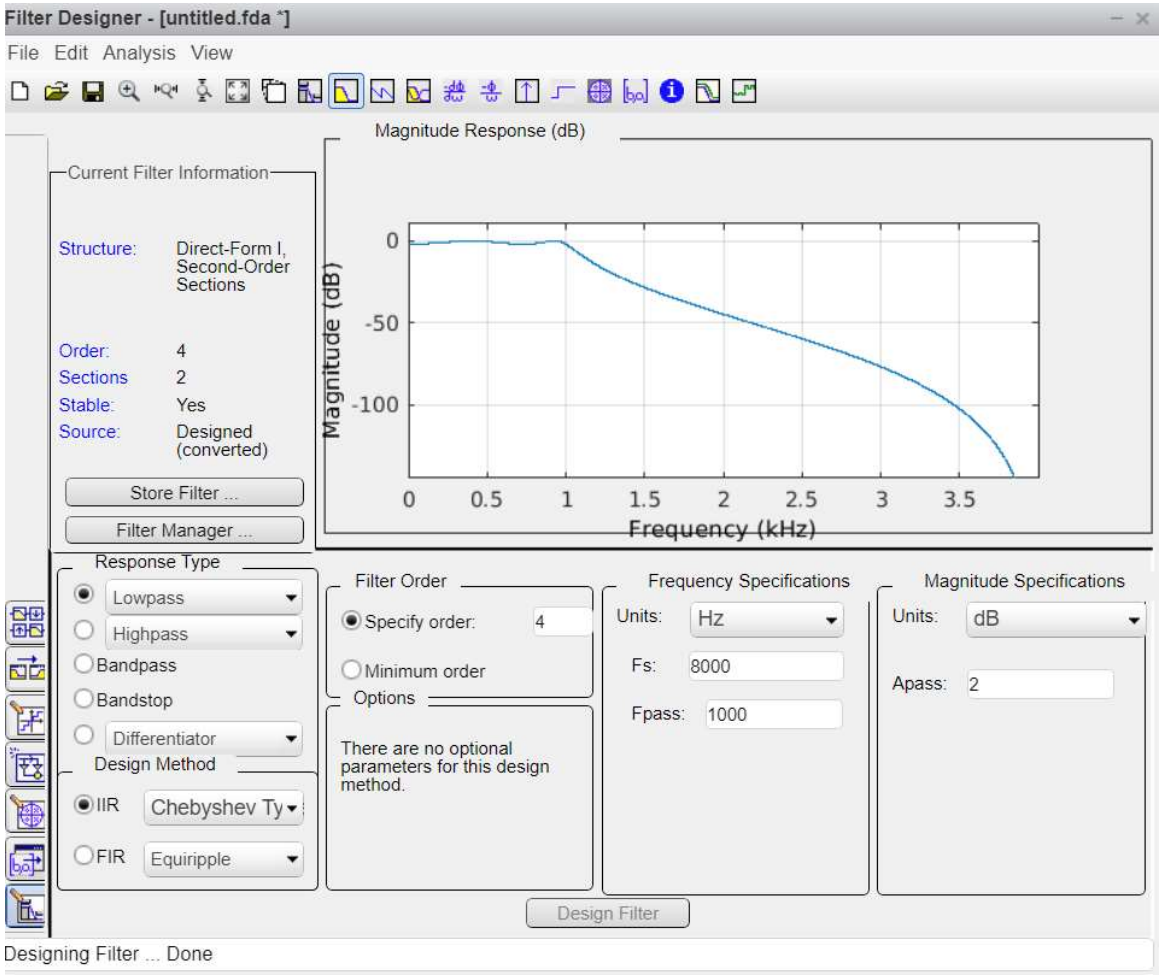
```
void arm_biquad_cascade_df1_f32 (  const arm_biquad_casd_df1_inst_f32 *  S,
                                   const float32_t *                     pSrc,
                                   float32_t *                           pDst,
                                   uint32_t                              blockSize
                                 )
```

Parameters

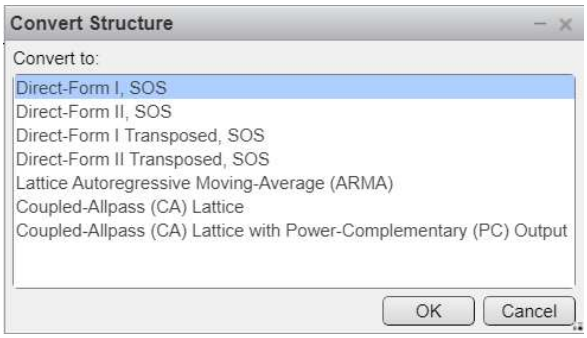
[in]	**S**	points to an instance of the floating-point Biquad cascade structure
[in]	**pSrc**	points to the block of input data
[out]	**pDst**	points to the block of output data
[in]	**blockSize**	number of samples to process

IIR filtro projektavimas su MATLAB *filterDesigner* įrankiu

Įrankio iškvietimas MATLAB aplinkoje: >> *filterDesigner*



Filtro struktūros parinkimas: *Edit -> Convert Structure...*



Eksportuojame filtro koeficientus: *File -> Export*
Dialogo lange pasirenkame: *Workspace, Coefficients, SOS, G*
Koeficientų išsaugojimui CMSIS bibliotekai reikiamu formatu naudojame paruoštą funkciją (skriptas faile stm32f4_iirsos_coeffs.m):

1. MATLAB komandinėje eilutėje įvedame: `stm32f4_iirsos_coeffs(SOS,G)`
2. Nurodome norimą failo pavadinimą, pavyzdžiui, *iir_coeffs_4ord.txt*

Įvykdžius nurodytus žingsnius, prieš tai suprojektuoto filtro į MATLAB kintamuosius eksportuoti koeficientai bus išsaugoti tekstiniame faile *"iir_coeffs_4ord.txt"*:

```
// iir_coeffs_4ord.txt
// this file was generated using
// function STM32F4_iirsos_coeffs.m

#define NUM_SECTIONS 2

float b[NUM_SECTIONS][3] = {
{1.27854304E-01, 2.55708609E-01, 1.27854304E-01},
{3.04659786E-02, 6.09319571E-02, 3.04659786E-02} };

float a[NUM_SECTIONS][3] = {
{1.00000000E+00, -1.34913555E+00, 8.60552772E-01},
{1.00000000E+00, -1.54190232E+00, 6.63766234E-01} };
```

MATLAB skriptas IIR filtro koeficientų eksportavimui (*failas stm32f4_iirsos_coeffs.m*[1])

```
% STM32F4_IIRSOS_COEFFS.M
%
% MATLAB function to write SOS IIR filter coefficients
% in format suitable for use in STM32F4 Discovery programs
% including stm32f4_iirsos_intr.c, stm32f4_iirsosprn_intr.c and
% stm32f4_iirsosdelta_intr.c
% assumes that coefficients have been exported from
% fdatool as two matrices
% first matrix has format
% [ b10 b11 b12 a10 a11 a12
%   b20 b21 b22 a20 a21 a22
%   ...
% ]
% where bij is the bj coefficient in the ith stage
% second matrix contains gains for each stage
%
function STM32F4_iirsos_coeffs(coeff,gain)
%
num_sections=length(gain)-1;
fname = input('enter filename for coefficients ','s');
fid = fopen(fname,'wt');
fprintf(fid,'// %s\n',fname);
fprintf(fid,'// this file was generated using');
fprintf(fid,'\n// function STM32F4_iirsos_coeffs.m\n',fname);
fprintf(fid,'\n#define NUM_SECTIONS %d\n',num_sections);
% first write the numerator coefficients b
% i is used to count through sections
fprintf(fid,'\nfloat b[NUM_SECTIONS][3] = { \n');
for i=1:num_sections
  if i==num_sections
    fprintf(fid,'{%2.8E, %2.8E, %2.8E} };\n',...
    coeff(i,1)*gain(i),coeff(i,2)*gain(i),coeff(i,3)*gain(i));
  else
    fprintf(fid,'{%2.8E, %2.8E, %2.8E},\n',...
```

[1] Donald S. Reay. Digital signal processing using the ARM Cortex-M4

```
    coeff(i,1)*gain(i),coeff(i,2)*gain(i),coeff(i,3)*gain(i));
  end
end
% then write the denominator coefficients a
% i is used to count through sections
fprintf(fid,'\nfloat a[NUM_SECTIONS][3] = { \n');
for i=1:num_sections
  if i==num_sections
    fprintf(fid,'{%2.8E, %2.8E, %2.8E} };\n',...
    coeff(i,4),coeff(i,5),coeff(i,6));
  else
    fprintf(fid,'{%2.8E, %2.8E, %2.8E},\n',...
    coeff(i,4),coeff(i,5),coeff(i,6));
  end
end
fclose(fid);
```

STM32F4 IIR programos pavyzdys

```
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under Ultimate Liberty license
  * SLA0044, the "License"; You may not use this file except in compliance with
  * the License. You may obtain a copy of the License at:
  *                              www.st.com/SLA0044
  *
  ******************************************************************************
  */
/* USER CODE END Header */

/* Includes ------------------------------------------------------------------*/
#include "main.h"
//#include "usb_host.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "arm_math.h"
#include "stm32f4_discovery_audio.h"

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define SAMPLING_FREQ 8000.0      //BSP_audio_out sampling frequency
```

```
#define TEST_LENGTH_SAMPLES       256
#define BLOCKSIZE                 32
#define NUMBLOCKS                 (TEST_LENGTH_SAMPLES/BLOCKSIZE)

#define NUM_SECTIONS              2

/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */
/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
I2C_HandleTypeDef hi2c1;
I2S_HandleTypeDef hi2s3;
DMA_HandleTypeDef hdma_spi3_tx;
SPI_HandleTypeDef hspi1;

/* USER CODE BEGIN PV */
static int16_t OutputBuffer[TEST_LENGTH_SAMPLES*2];     //Output, left+right channels
static int16_t random_temp;

            //IIR related PV
static float32_t testInput_f32[TEST_LENGTH_SAMPLES];    //IIR input signal buffer
static float32_t testOutput_f32[TEST_LENGTH_SAMPLES];   //IIR output signal buffer
float32_t *inputF32 = &testInput_f32[0];
float32_t *outputF32 = &testOutput_f32[0];

float32_t coeffs[5*NUM_SECTIONS] = {0};
float32_t state[4*NUM_SECTIONS] = {0};

            //IIR filter coeffs
float b[NUM_SECTIONS][3] = {
{1.27854304E-01, 2.55708609E-01, 1.27854304E-01},
{3.04659786E-02, 6.09319571E-02, 3.04659786E-02} };

float a[NUM_SECTIONS][3] = {
{1.00000000E+00, -1.34913555E+00, 8.60552772E-01},
{1.00000000E+00, -1.54190232E+00, 6.63766234E-01} };


/* USER CODE END PV */

/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_I2C1_Init(void);
static void MX_I2S3_Init(void);
static void MX_SPI1_Init(void);
//void MX_USB_HOST_Process(void);

/* USER CODE BEGIN PFP */

/* USER CODE END PFP */

/* Private user code ---------------------------------------------------------*/
/* USER CODE BEGIN 0 */
```

```
            //White noise generator for noise testing
            int32_t rand_int(void)
            {
                        static int32_t a = 100001;
                        a = (a*125) % 2796203;
                        return a;
            }

/* USER CODE END 0 */

/**
  * @brief  The application entry point.
  * @retval int
  */
int main(void)
{
  /* USER CODE BEGIN 1 */
            //--- IIR structure ---
            arm_biquad_casd_df1_inst_f32 S;

            //--- write IIR coeffs to the CMSIS compatible array
            uint16_t i,k=0;
            for (i=0; i<NUM_SECTIONS ; i++)
            {
                        coeffs[k++] = b[i][0];
                        coeffs[k++] = b[i][1];
                        coeffs[k++] = b[i][2];
                        coeffs[k++] = -a[i][1];
                        coeffs[k++] = -a[i][2];
            }

  /* USER CODE END 1 */

  /* MCU Configuration--------------------------------------------------------*/

  /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
  HAL_Init();

  /* USER CODE BEGIN Init */
  /* USER CODE END Init */

  /* Configure the system clock */
  SystemClock_Config();

  /* USER CODE BEGIN SysInit */
  /* USER CODE END SysInit */

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_I2C1_Init();
  MX_I2S3_Init();
  MX_SPI1_Init();
  //MX_USB_HOST_Init();
  /* USER CODE BEGIN 2 */

       //--- Call IIR init function
       arm_biquad_cascade_df1_init_f32(&S, NUM_SECTIONS, &(coeffs[0]), &(state[0]));

      /*** Initialize the Audio codec (I2S, I2C, IOExpander, IOs...) ***/
```

```
        if(BSP_AUDIO_OUT_Init(OUTPUT_DEVICE_AUTO, 80, SAMPLING_FREQ) != AUDIO_OK)
        {
            Error_Handler();
        }

      /*** Test Signal generation ***/
        for(int index = 0; index < TEST_LENGTH_SAMPLES; index++)
        {
            //Test signal 1000 Hz + 3000 Hz
            testInput_f32[index] = ( 30000*sin(32*2*PI*index/TEST_LENGTH_SAMPLES) +
15000*sin(96*2*PI*index/TEST_LENGTH_SAMPLES));
        }

       /*** Perform IIR filtering operation ***/
          for (k=0; k < NUMBLOCKS; k++)
          {
              arm_biquad_cascade_df1_f32 (&S, inputF32 + (k*BLOCKSIZE), outputF32 +
(k*BLOCKSIZE), BLOCKSIZE);
          }

       /*** Fill Output Buffer ***/
          for(int i = 0; i < TEST_LENGTH_SAMPLES; i++)
          {
              OutputBuffer[i<<1] = (int16_t)testInput_f32[i];
              OutputBuffer[(i<<1)+1] = (int16_t)testOutput_f32[i];
          }

       /*** Start Audio play ***/
          BSP_AUDIO_OUT_Play((uint16_t*)&OutputBuffer[0], TEST_LENGTH_SAMPLES*2);

  /* USER CODE END 2 */

  /* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
              HAL_GPIO_TogglePin(LD4_GPIO_Port, LD4_Pin);
              HAL_Delay(250);
    /* USER CODE END WHILE */
    //MX_USB_HOST_Process();

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}

/**
  * @brief System Clock Configuration
  * @retval None
  */
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
  RCC_PeriphCLKInitTypeDef PeriphClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
```

```
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
  RCC_OscInitStruct.HSEState = RCC_HSE_ON;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
  RCC_OscInitStruct.PLL.PLLM = 8;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
  /** Initializes the CPU, AHB and APB busses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5) != HAL_OK)
  {
    Error_Handler();
  }
  PeriphClkInitStruct.PeriphClockSelection = RCC_PERIPHCLK_I2S;
  PeriphClkInitStruct.PLLI2S.PLLI2SN = 192;
  PeriphClkInitStruct.PLLI2S.PLLI2SR = 2;
  if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInitStruct) != HAL_OK)
  {
    Error_Handler();
  }
}

/**
  * @brief I2C1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2C1_Init(void)
{

  /* USER CODE BEGIN I2C1_Init 0 */
  /* USER CODE END I2C1_Init 0 */

  /* USER CODE BEGIN I2C1_Init 1 */
  /* USER CODE END I2C1_Init 1 */
  hi2c1.Instance = I2C1;
  hi2c1.Init.ClockSpeed = 100000;
  hi2c1.Init.DutyCycle = I2C_DUTYCYCLE_2;
  hi2c1.Init.OwnAddress1 = 0;
  hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
  hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
  hi2c1.Init.OwnAddress2 = 0;
  hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
  hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
  if (HAL_I2C_Init(&hi2c1) != HAL_OK)
  {
```

```
    Error_Handler();
  }
  /* USER CODE BEGIN I2C1_Init 2 */
  /* USER CODE END I2C1_Init 2 */
}

/**
  * @brief I2S3 Initialization Function
  * @param None
  * @retval None
  */
static void MX_I2S3_Init(void)
{

  /* USER CODE BEGIN I2S3_Init 0 */
  /* USER CODE END I2S3_Init 0 */

  /* USER CODE BEGIN I2S3_Init 1 */
  /* USER CODE END I2S3_Init 1 */
  hi2s3.Instance = SPI3;
  hi2s3.Init.Mode = I2S_MODE_MASTER_TX;
  hi2s3.Init.Standard = I2S_STANDARD_PHILIPS;
  hi2s3.Init.DataFormat = I2S_DATAFORMAT_16B;
  hi2s3.Init.MCLKOutput = I2S_MCLKOUTPUT_ENABLE;
  hi2s3.Init.AudioFreq = I2S_AUDIOFREQ_96K;
  hi2s3.Init.CPOL = I2S_CPOL_LOW;
  hi2s3.Init.ClockSource = I2S_CLOCK_PLL;
  hi2s3.Init.FullDuplexMode = I2S_FULLDUPLEXMODE_DISABLE;
  if (HAL_I2S_Init(&hi2s3) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN I2S3_Init 2 */
  /* USER CODE END I2S3_Init 2 */
}

/**
  * @brief SPI1 Initialization Function
  * @param None
  * @retval None
  */
static void MX_SPI1_Init(void)
{

  /* USER CODE BEGIN SPI1_Init 0 */
  /* USER CODE END SPI1_Init 0 */

  /* USER CODE BEGIN SPI1_Init 1 */
  /* USER CODE END SPI1_Init 1 */
  /* SPI1 parameter configuration*/
  hspi1.Instance = SPI1;
  hspi1.Init.Mode = SPI_MODE_MASTER;
  hspi1.Init.Direction = SPI_DIRECTION_2LINES;
  hspi1.Init.DataSize = SPI_DATASIZE_8BIT;
  hspi1.Init.CLKPolarity = SPI_POLARITY_LOW;
  hspi1.Init.CLKPhase = SPI_PHASE_1EDGE;
  hspi1.Init.NSS = SPI_NSS_SOFT;
  hspi1.Init.BaudRatePrescaler = SPI_BAUDRATEPRESCALER_2;
  hspi1.Init.FirstBit = SPI_FIRSTBIT_MSB;
  hspi1.Init.TIMode = SPI_TIMODE_DISABLE;
```

```
  hspi1.Init.CRCCalculation = SPI_CRCCALCULATION_DISABLE;
  hspi1.Init.CRCPolynomial = 10;
  if (HAL_SPI_Init(&hspi1) != HAL_OK)
  {
    Error_Handler();
  }
  /* USER CODE BEGIN SPI1_Init 2 */
  /* USER CODE END SPI1_Init 2 */
}

/**
  * Enable DMA controller clock
  */
static void MX_DMA_Init(void)
{

  /* DMA controller clock enable */
  __HAL_RCC_DMA1_CLK_ENABLE();

  /* DMA interrupt init */
  /* DMA1_Stream5_IRQn interrupt configuration */
  HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);

}

/**
  * @brief GPIO Initialization Function
  * @param None
  * @retval None
  */
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};

  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOE_CLK_ENABLE();
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  __HAL_RCC_GPIOD_CLK_ENABLE();

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(CS_I2C_SPI_GPIO_Port, CS_I2C_SPI_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(OTG_FS_PowerSwitchOn_GPIO_Port, OTG_FS_PowerSwitchOn_Pin,
GPIO_PIN_SET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOD, LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                          |Audio_RST_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : CS_I2C_SPI_Pin */
  GPIO_InitStruct.Pin = CS_I2C_SPI_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(CS_I2C_SPI_GPIO_Port, &GPIO_InitStruct);
```

```
  /*Configure GPIO pin : OTG_FS_PowerSwitchOn_Pin */
  GPIO_InitStruct.Pin = OTG_FS_PowerSwitchOn_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(OTG_FS_PowerSwitchOn_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : PDM_OUT_Pin */
  GPIO_InitStruct.Pin = PDM_OUT_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
  HAL_GPIO_Init(PDM_OUT_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : B1_Pin */
  GPIO_InitStruct.Pin = B1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(B1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : BOOT1_Pin */
  GPIO_InitStruct.Pin = BOOT1_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(BOOT1_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : CLK_IN_Pin */
  GPIO_InitStruct.Pin = CLK_IN_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  GPIO_InitStruct.Alternate = GPIO_AF5_SPI2;
  HAL_GPIO_Init(CLK_IN_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LD4_Pin LD3_Pin LD5_Pin LD6_Pin
                           Audio_RST_Pin */
  GPIO_InitStruct.Pin = LD4_Pin|LD3_Pin|LD5_Pin|LD6_Pin
                        |Audio_RST_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOD, &GPIO_InitStruct);

  /*Configure GPIO pin : OTG_FS_OverCurrent_Pin */
  GPIO_InitStruct.Pin = OTG_FS_OverCurrent_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(OTG_FS_OverCurrent_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pin : MEMS_INT2_Pin */
  GPIO_InitStruct.Pin = MEMS_INT2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_EVT_RISING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(MEMS_INT2_GPIO_Port, &GPIO_InitStruct);

}

/* USER CODE BEGIN 4 */
```

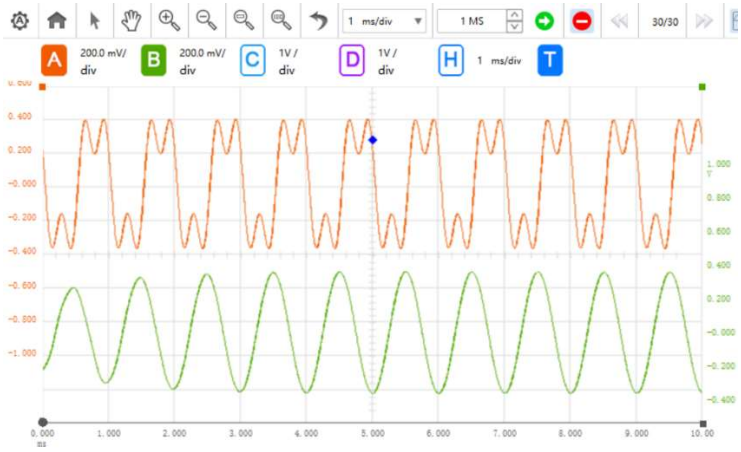
```
      /*** Back to Buffer beginning ***/
      void BSP_AUDIO_OUT_TransferComplete_CallBack(void)
      {
          BSP_AUDIO_OUT_ChangeBuffer((uint16_t*)&OutputBuffer[0], TEST_LENGTH_SAMPLES*2);
          HAL_GPIO_TogglePin(LD3_GPIO_Port, LD3_Pin);
      }

/* USER CODE END 4 */

/**
  * @brief  This function is executed in case of error occurrence.
  * @retval None
  */
void Error_Handler(void)
{
  /* USER CODE BEGIN Error_Handler_Debug */
  /* User can add his own implementation to report the HAL error return state */
  /* USER CODE END Error_Handler_Debug */
}

#ifdef  USE_FULL_ASSERT
/**
  * @brief  Reports the name of the source file and the source line number
  *         where the assert_param error has occurred.
  * @param  file: pointer to the source file name
  * @param  line: assert_param error line source number
  * @retval None
  */
void assert_failed(uint8_t *file, uint32_t line)
{
  /* USER CODE BEGIN 6 */
  /* User can add his own implementation to report the file name and line number,
     tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
  /* USER CODE END 6 */
}
#endif /* USE_FULL_ASSERT */

    /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
```



Praktinė dalis

- Suprojektuoti žemo dažnio, juostinius bei aukšto dažnio IIR filtrus naudojant MATLAB FilterDesigner įrankį. Eksportuoti filtro koeficientus.
- Išbandyti suprojektuotų filtrų veikimą STM32F407 mikrovaldikliu, testavimui naudojant poli-harmoninį signalą.
- Išbandyti suprojektuotų filtrų impulsinę charakteristiką.