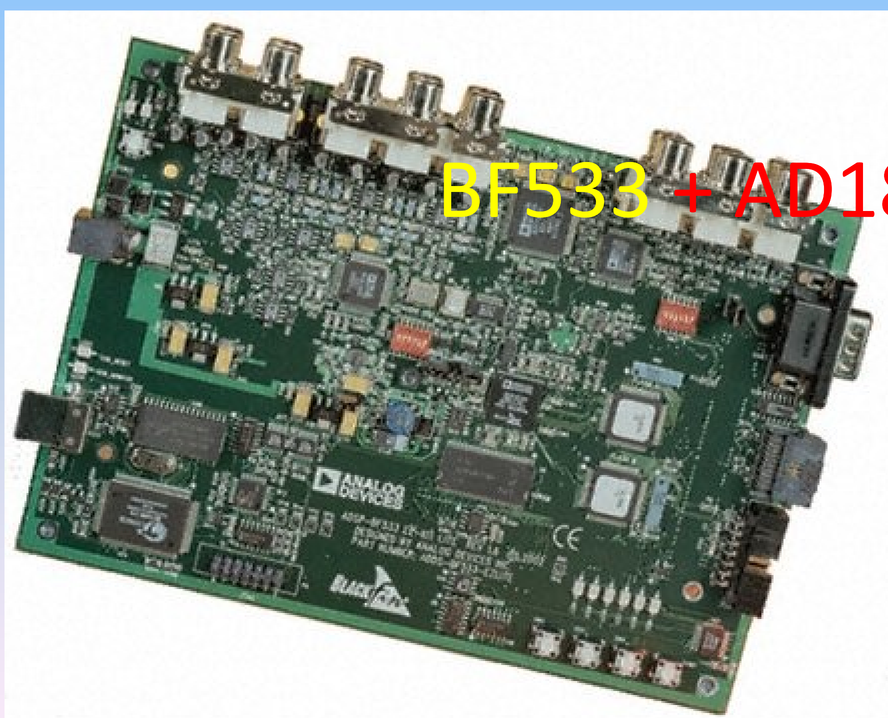
DSP

Lab. 3 – Audio signal processing with the Blackfin BF533 DSP

Stefan Ataman

2013 – 2014

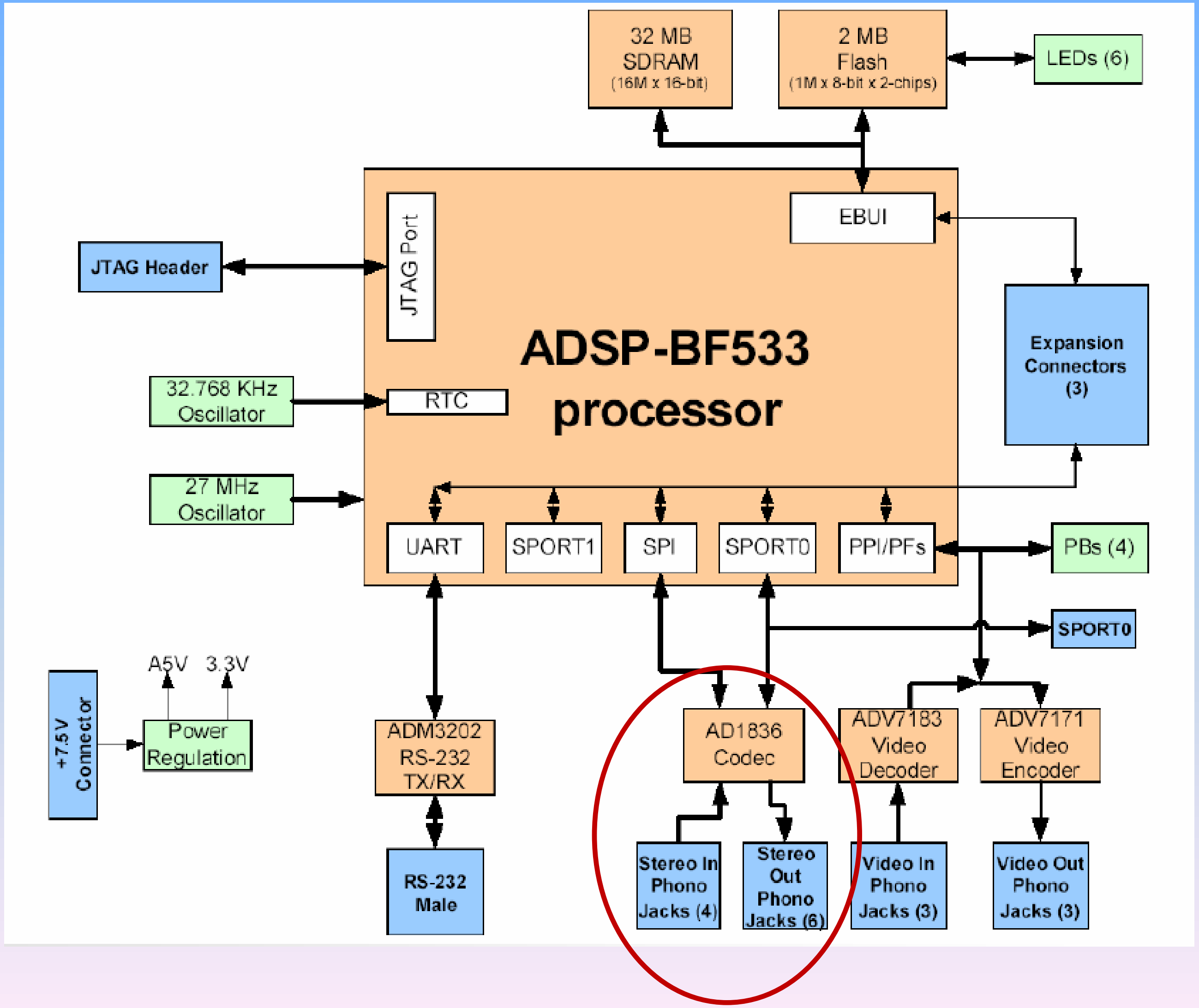
Blackfin development
using the BF533 EZ-KIT Lite



BF533 + AD1836 audio codec

ADSP-BF533 EZ-KIT Lite

real-time audio processing



In this lab we want to:

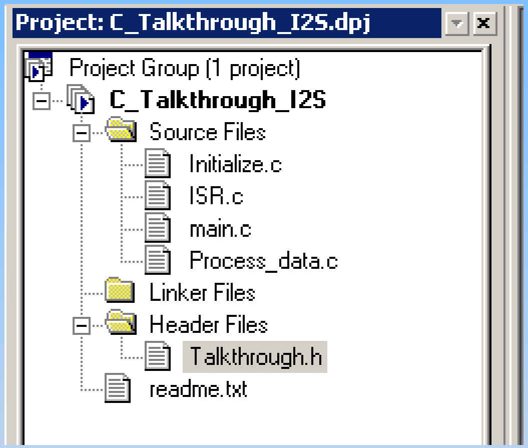
- 1) open the talk-through FIR project
- 2) activate a FIR filter
- 3) implement low-pass, high-pass, band-pass, band-stop filters

But before…

… you have a set of questions concerning the previous lab!

Question set 3:

- explain in brief how the talkthrough project worked
  - the project had the structure:



- where do you add your data processing algorithm?

Question set 3:

- what does the ISR.c file contain?

- how come that your code was working but in the main loop you had a **`while(1)`** instruction?

```
void main(void)
{

    sysreg_write(reg_SYSCFG, 0x32);
    Init_EBIU();
    Init_Flash();
    Init1836();
    Init_Sport0();

    Init_FIR();

    Init_DMA();
    Init_Interrupts();
    Enable_DMA_Sport0();


    while(1);

}
```

1. Open the talk-through C project in Visual DSP++ IDE

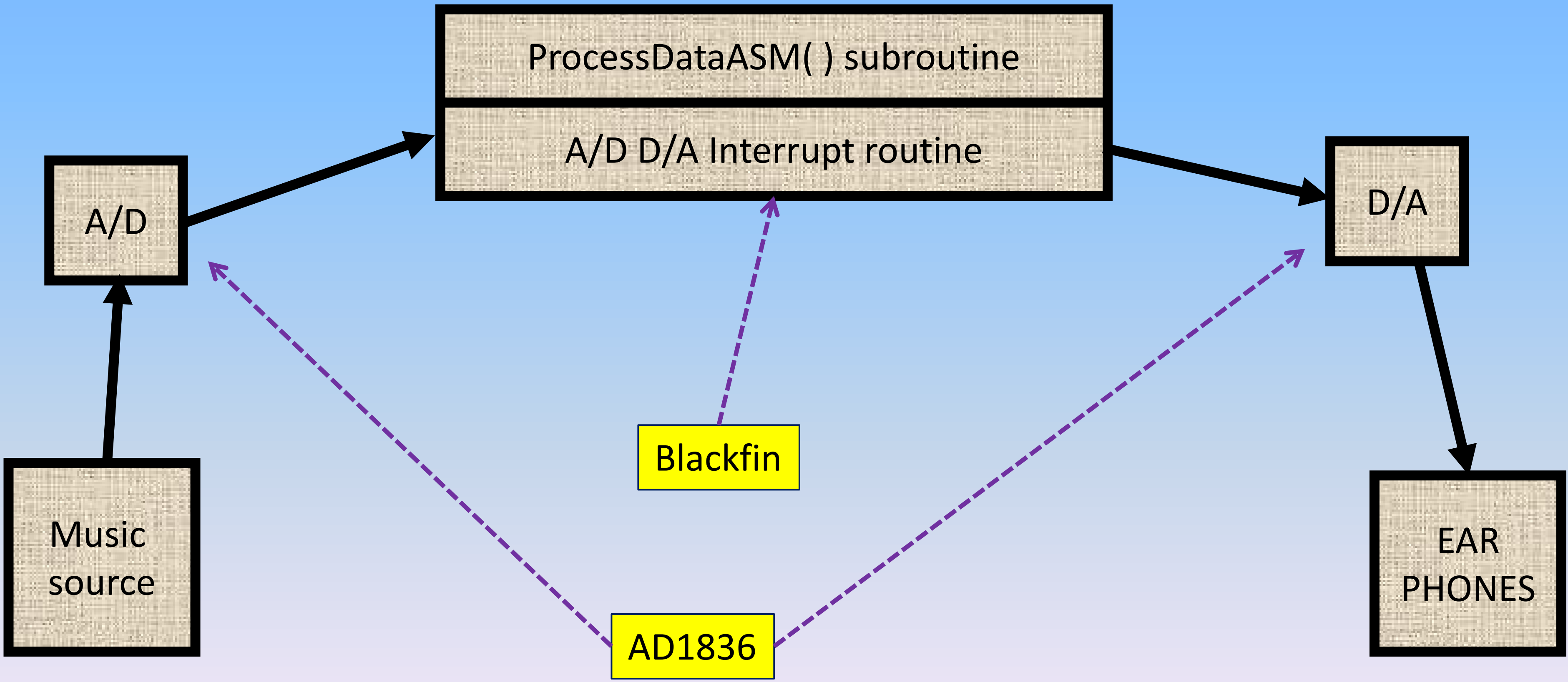
C Talk-through I^2S FIR

Open a Project

- From the File menu, choose Open and then Project
  - VisualDSP++ displays the Open Project dialog box.
- From campus download the archive :
  - C_talkthrough_i2s_fir16.zip
- Unpack it to your own folder, *e.g.*
  - d:\your_name_GrXX\C_talkthrough_I2S_FIR

Hardware reminder

- data path:



Audio signal processing

implementing a FIR filter

The FIR filter

how to implement a FIR filter on a Blackfin?

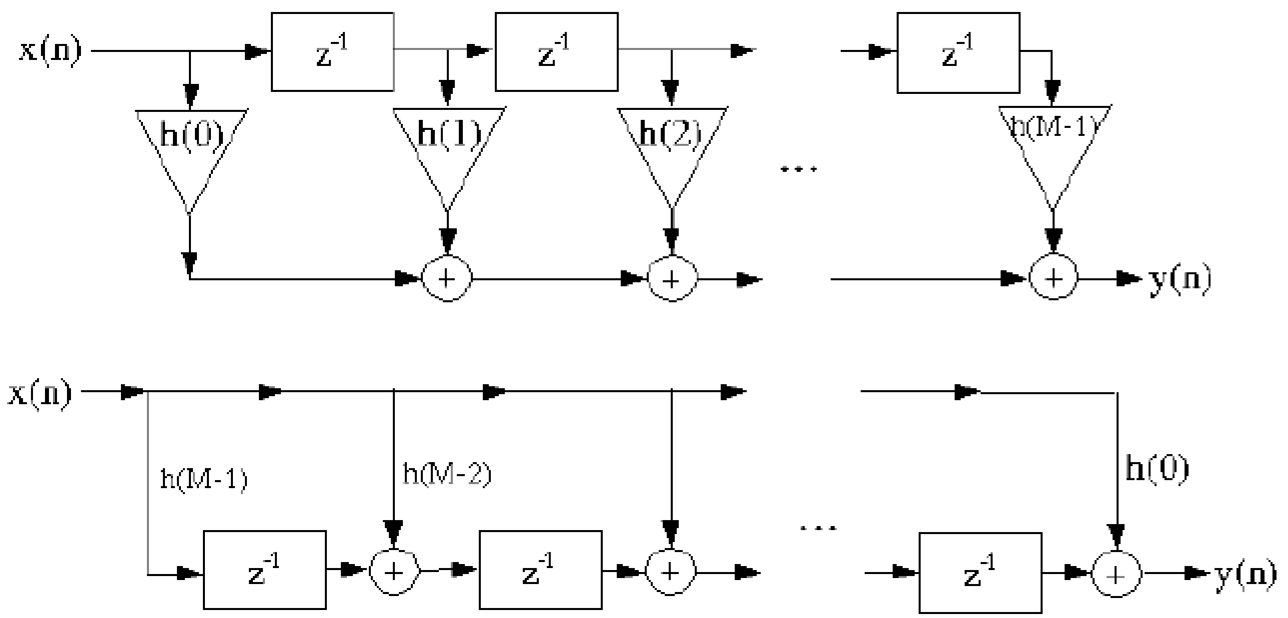
What is a FIR filter anyway?

Reminder:

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

Direct-form

Transposition of direct-form



How to implement it?

- cheer up: Analog Devices did it already for you!
- include **`filter.h`**
- use the function **`fir_fr16`** to filter
- user the function **`fir_init`** to init the filter

Use the FIR Function from the Blackfin DSP Library

- in talkthrough.h you have `#include <filter.h>`
- the filter input arguments are:

```
fract16 x[];        /* Input sample vector x */
fract16 y[];        /* Output sample vector y */
int inum;           /* Number of input samples */
fir_state_fr16 *state;
      /* Pointer to filter state structure */
```

Use the FIR Function from the Blackfin DSP Library

- how to init the function:

```
int main() {
…
fir_init(state, coeffs_array, delayline,
  num_coeffs, interpolation_idx);
}
```

Use the FIR Function from the Blackfin DSP Library

- how to call it:

```
fir_fr16(x, y, inum, state) ;
```

input signal sample

output signal sample

number of taps
(filter length)

filter internal state

state is a structure (see next slide)

Use the FIR Function from the Blackfin DSP Library

- the FIR state structure is defined as:

```
typedef struct
{
fract16 *h,     /* filter coefficients */
fract16 *d,     /* start of delay line */
fract16 *p,     /* read/write pointer */
int k;          /* number of coefficients */
int l; /* interpolation/decimation index */
} fir_state_fr16;
```

Let's implement now

a FIR filter in the talkthrough demo

Note

- everything is DONE, all you have to do is UNDERSTAND!

- you will have to create the FIR coefficients (later)

In talkthrough.h

- are declared the number of taps of your FIR filter (see next slide)

- are declared the FIR related variables and structures

(see next slides for details)

Declare the number of taps

Talkthrough.h *

```
// SPORT0 word length
#define SLEN_24 0x0017
#define SLEN_32 0x001f
#define SLEN_16 0x000f

// DMA flow mode
#define FLOW_1  0x1000

// Audio Buffer Size
#define BUFFER_SIZE 128

// FIR number of TAPS - define it here
#define TAPS    32


//--------------------------------------------------------------------------//
// Global variables                                                         //
//--------------------------------------------------------------------------//
extern short sChannel0LeftIn[];
extern short sChannel0RightIn[];
extern short sChannel0LeftOut[];
extern short sChannel0RightOut[];

//FIR filter
extern fract16 delayL[];
extern fract16 delayR[];
extern fract16 coef[];
```

This time channel variables are of short (int) type

Talkthrough.h *

```
//--------------------------------------------------------------------------//
// Global variables                                                         //
//--------------------------------------------------------------------------//
extern short sChannel0LeftIn[];
extern short sChannel0RightIn[];
extern short sChannel0LeftOut[];
extern short sChannel0RightOut[];

//FIR filter
extern fract16 delayL[];
extern fract16 delayR[];
extern fract16 coef[];
extern fir_state_fr16 stateL;
extern fir_state_fr16 stateR;


extern volatile short sCodec1836TxRegs[];
extern volatile short sRxBuffer[];
extern volatile short sTxBuffer[];

//--------------------------------------------------------------------------//
// Prototypes                                                               //
//--------------------------------------------------------------------------//
// in file Initialize.c
```

The channel variables (vectors) are

```
// left input data from ad1836
short sChannel0LeftIn[BUFFER_SIZE];
// right input data from ad1836
short sChannel0RightIn[BUFFER_SIZE];
// left output data for ad1836
short sChannel0LeftOut[BUFFER_SIZE];
// right output data for ad1836
short sChannel0RightOut[BUFFER_SIZE];
```

Buffer size has also been declared

Talkthrough.h *

```
// SPORT0 word length
#define SLEN_24 0x0017
#define SLEN_32 0x001f
#define SLEN_16 0x000f

// DMA flow mode
#define FLOW_1  0x1000

// Audio Buffer Size
#define BUFFER_SIZE 128

// FIR number of TAPS - define it here
#define TAPS    32


//--------------------------------------------------------------------------//
// Global variables                                                          //
//--------------------------------------------------------------------------//
extern short sChannel0LeftIn[];
extern short sChannel0RightIn[];
extern short sChannel0LeftOut[];
extern short sChannel0RightOut[];

//FIR filter
extern fract16 delayL[];
extern fract16 delayR[];
extern fract16 coef[];
```

Declare the FIR filters (two of them, LEFT and RIGHT)

Talkthrough.h *

```
//--------------------------------------------------------------------------//
// Global variables                                                         //
//--------------------------------------------------------------------------//
extern short sChannel0LeftIn[];
extern short sChannel0RightIn[];
extern short sChannel0LeftOut[];
extern short sChannel0RightOut[];

// the FIR filter related variables and structures
extern fract16 delayL[];
extern fract16 delayR[];
extern fract16 coef[];
extern fir_state_fr16 stateL;
extern fir_state_fr16 stateR;


extern volatile short sCodec1836TxRegs[];
extern volatile short sRxBuffer[];
extern volatile short sTxBuffer[];

//--------------------------------------------------------------------------//
// Prototypes                                                               //
//--------------------------------------------------------------------------//
// in file Initialize.c
```

=> you will call it in Process_Data.c

Init the delayL and delayR vectors

```
fract16 delayL[TAPS]={0};

fract16 delayR[TAPS]={0};
```

Init the FIR filters

Initialize.c

```
//------------------------------------------------------------------------------//
void Init_Flash(void)
{
    *pFlashA_PortA_Dir = 0x1;
}

//------------------------------------------------------------------------------//
// Function:    Init_FIR                                                        //
//                                                                              //
// Description: This function initializes FIR filter states for right & left//
//              channels                                                        //
//------------------------------------------------------------------------------//
void Init_FIR(void)
{
    fir_init(stateR, coef, delayR, TAPS, 1);
    fir_init(stateL, coef, delayL, TAPS, 1);
}


//------------------------------------------------------------------------------//
// Function:    Init1836()                                                      //
//                                                                              //
// Description: This function sets up the SPI port to configure the AD1836. //
//              The content of the array sCodec1836TxRegs is sent to the       //
//              codec.                                                          //
//------------------------------------------------------------------------------//
void Init1836(void)
```

Init the FIR filters

```
void Init_FIR(void){

    fir_init(stateR, coef, delayR, TAPS, 1);

    fir_init(stateL, coef, delayL, TAPS, 1);

}
```

right filter internal state

filter coefficients

right filter delay vector

number of filter coefficients

left filter internal state

The throughput – no filter

Process_data.c

```
//                iChannel0RightIn, iChannel1LeftIn and iChannel1RightIn       //
//                respectively. The processed data should be stored in         //
//                iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut,        //
//                iChannel1RightOut, iChannel2LeftOut and iChannel2RightOut    //
//                respectively.                                                //
//--------------------------------------------------------------------------//
void Process_Data(void)
{
    int i;

    // Comment lines below as required (talkthrough or filterin)


    // Process loopback

    for (i=0; i<BUFFER_SIZE; i++)
    {
        sChannel0RightOut[i] = sChannel0RightIn[i];
        sChannel0LeftOut[i] = sChannel0LeftIn[i];
    }


    // Or process FIR filter
    //fir_fr16(sChannel0RightIn, sChannel0RightOut, BUFFER_SIZE, &stateR);
    //fir_fr16(sChannel0LeftIn, sChannel0LeftOut, BUFFER_SIZE, &stateL);
}
```

The throughput – no filter

- boring but check nonetheless that the audio connections are OK
  - build the project
  - run it
  - check if sound is OK

FIR filtering

Process_data.c *

```
//                iChannel0LeftOut, iChannel0RightOut, iChannel1LeftOut,        //
//                iChannel1RightOut, iChannel2LeftOut and iChannel2RightOut     //
//                respectively.                                                 //
//------------------------------------------------------------------------------//
void Process_Data(void)
{
    int i;

    // Comment lines below as required (talkthrough or filterin)


    // Process loopback

    for (i=0; i<BUFFER_SIZE; i++)
    /*{
        sChannel0RightOut[i] = sChannel0RightIn[i];
        sChannel0LeftOut[i] = sChannel0LeftIn[i];
    }
    */

    // Or process FIR filter
    fir_fr16(sChannel0RightIn, sChannel0RightOut, BUFFER_SIZE, &stateR);


    fir_fr16(sChannel0LeftIn, sChannel0LeftOut, BUFFER_SIZE, &stateL);
}
```

The FIR filtering is done by:

```
fir_fr16(sChannel0RightIn,
  sChannel0RightOut, BUFFER_SIZE, &stateR);
```

Right channel0 filter

```
fir_fr16(sChannel0LeftIn, sChannel0LeftOut,
  BUFFER_SIZE, &stateL);
```

Left channel0 filter

Question set #5

- explain the input variables (parameters) taken by the **`fir_fr16()`** function for:
  - the LEFT channel 0
  - the RIGHT channel 0

Where do the FIR filter

coefficients come from?

An external (*.dat) file:

main.c *

```c
// SPORT0 DMA receive buffer
volatile short sRxBuffer[2];



fract16 delayL[TAPS]={0};
fract16 delayR[TAPS]={0};

fract16 coef[TAPS] = {
    #include "coef32.dat"
    //#include "coef32_low1.dat"
    //#include "coef32_pass1.dat"
    //#include "coef32_high1.dat"
    //#include "coef32_pass2_341taps.dat"
    //#include "coef32_low2_158taps.dat"

};


fir_state_fr16 stateL;
fir_state_fr16 stateR;




//--------------------------------------------------------------------------//
// Function:    main                                                        //
```

And who put those coefficients

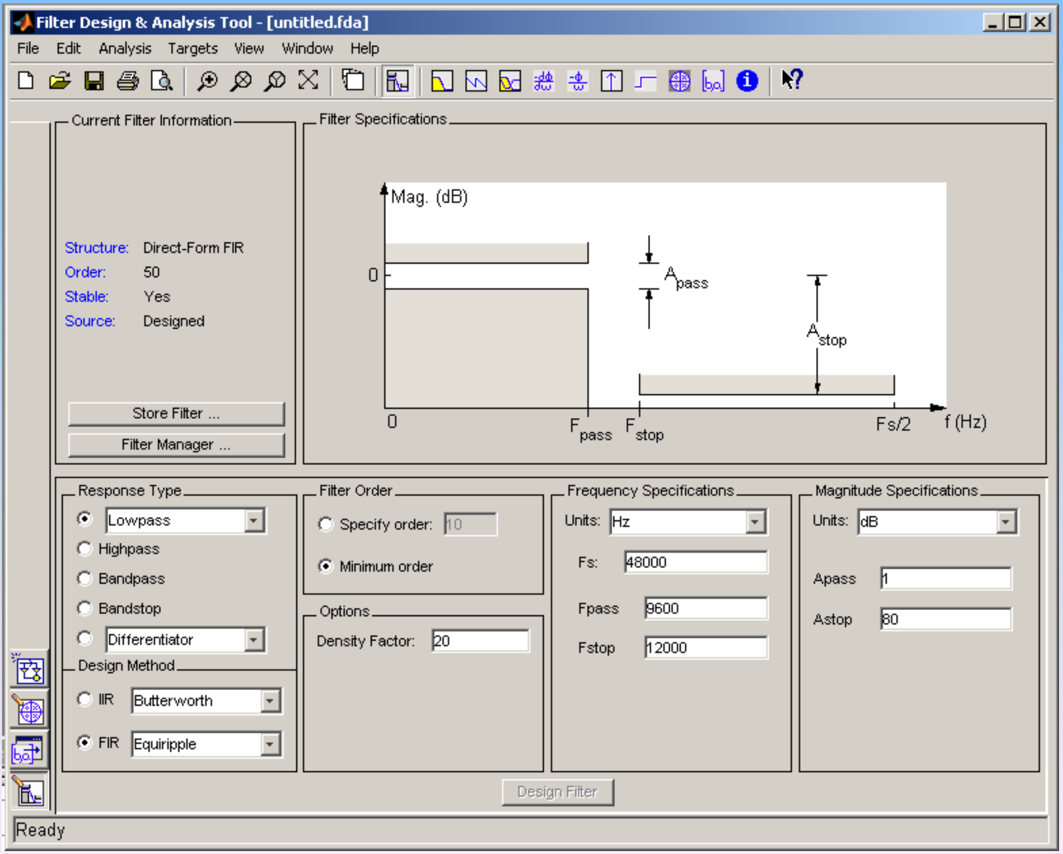
into the *.dat file?

Matlab's FDAtool!

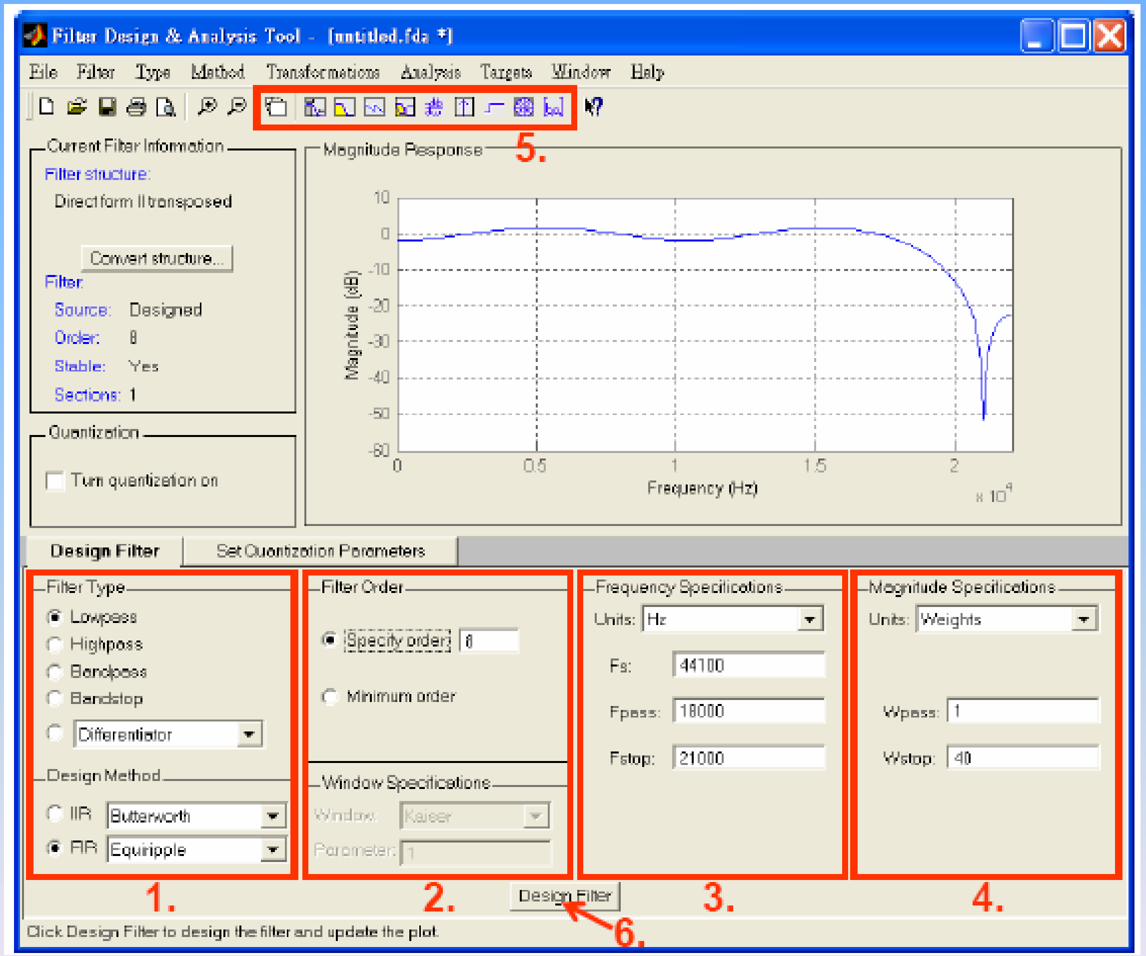
Filter design, the graphical way

Open Matlab

- type: **fdatool**
- a window opens:



You have the fields:



(see next slide)

You have the fields:

1: Filter Type:

- FIR or IIR, Low-pass or Band-pass…

2: Filter Order:

- 7, 8 or more…

3: Frequency Specification

- Fs, Fpass, Fstop…

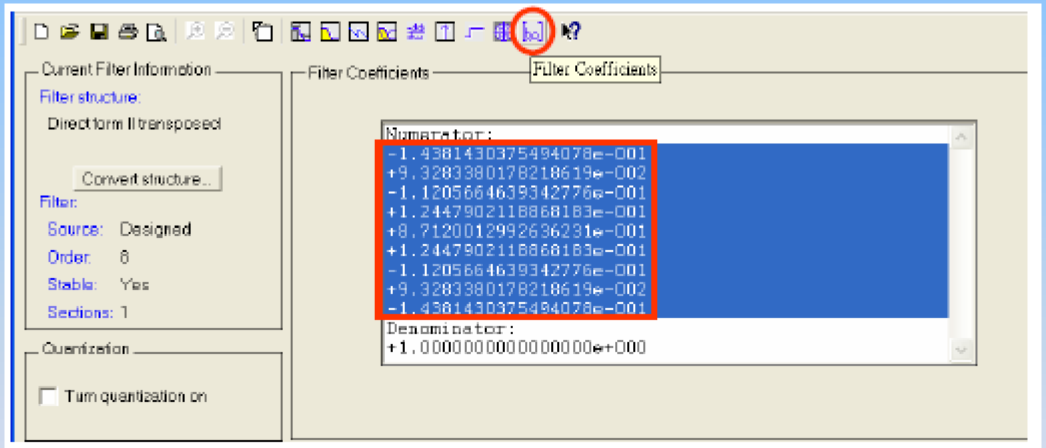
4: Magnitude Specification

- passband, stopband attenuation

You have the fields:

5: Magnitude response, graphical view

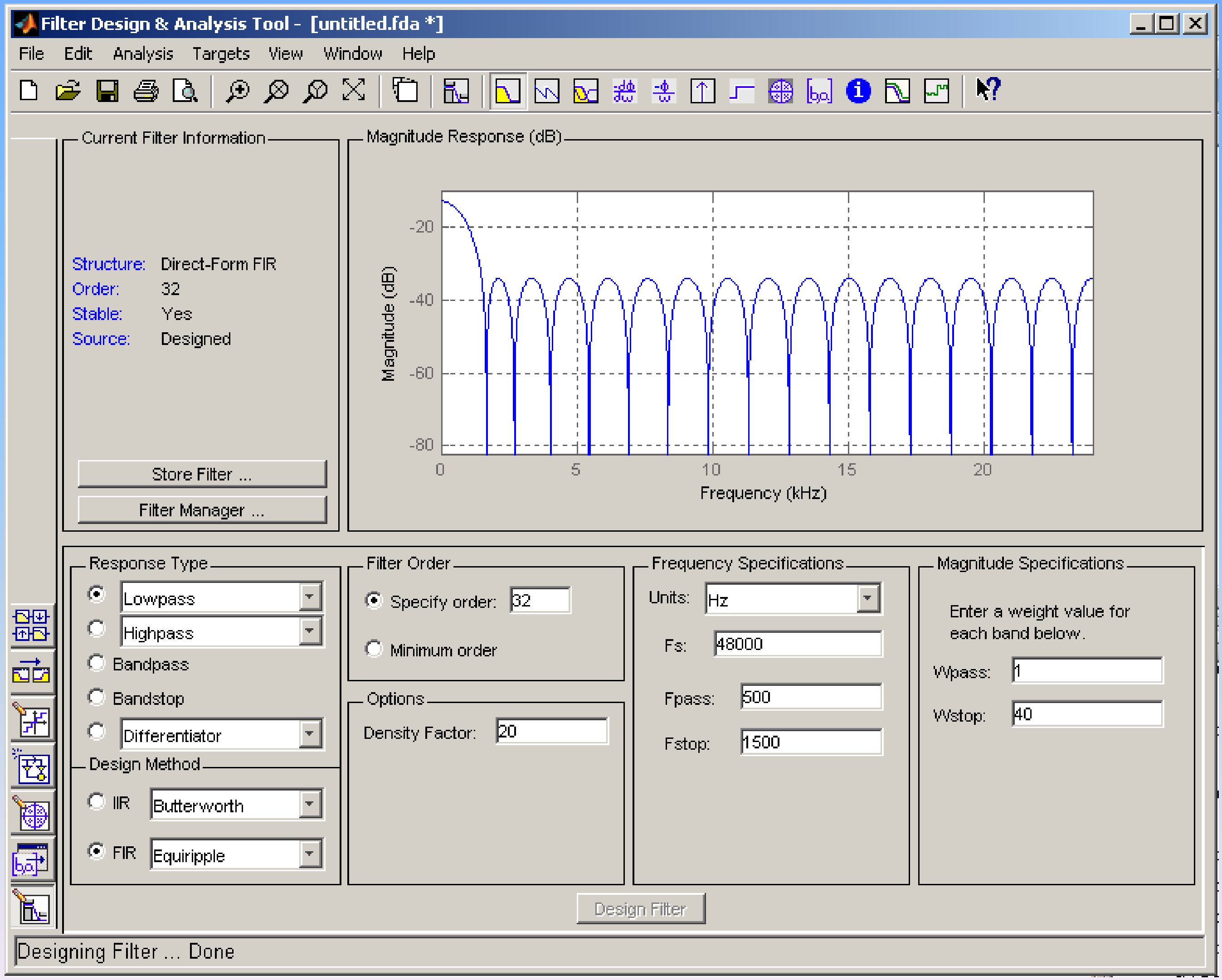
6: Design the Filter



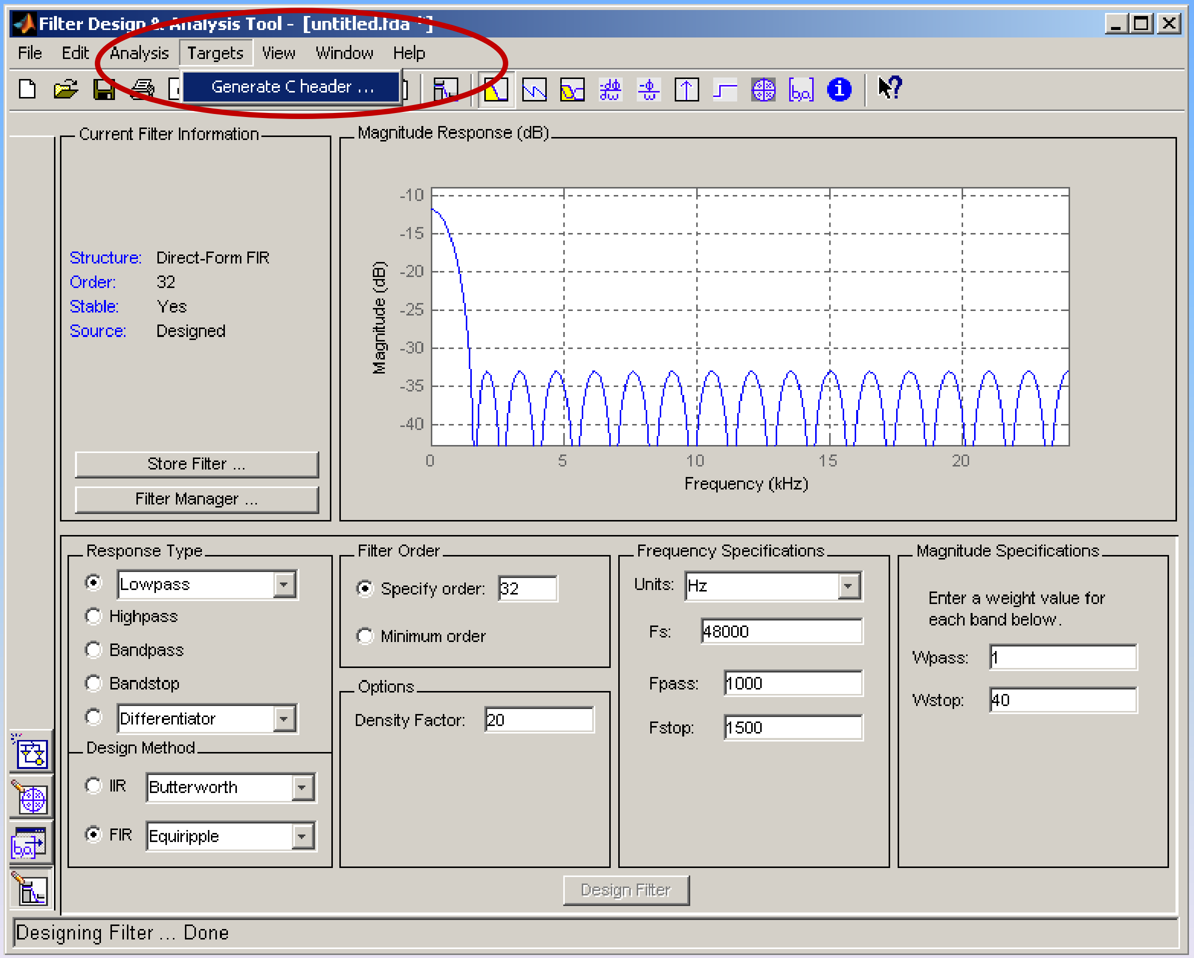
Task 1

- Design a low-pass filter:
  - FIR equiriple
  - Fpass=500
  - Fstop=1500
  - Filter order 32
  - Wpass=1
  - Wstop=40
- Click on "design filter"

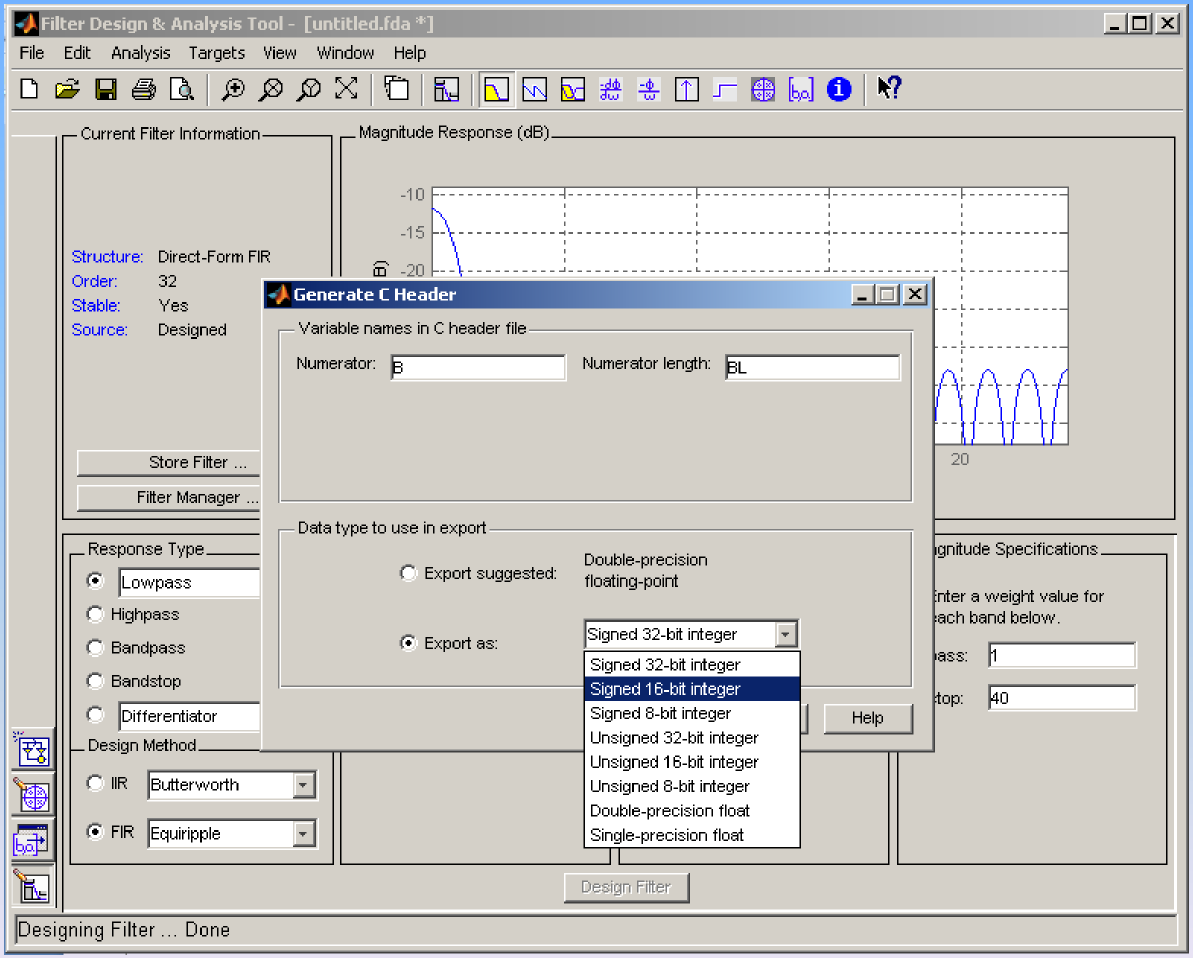
You obtain with FDAtool



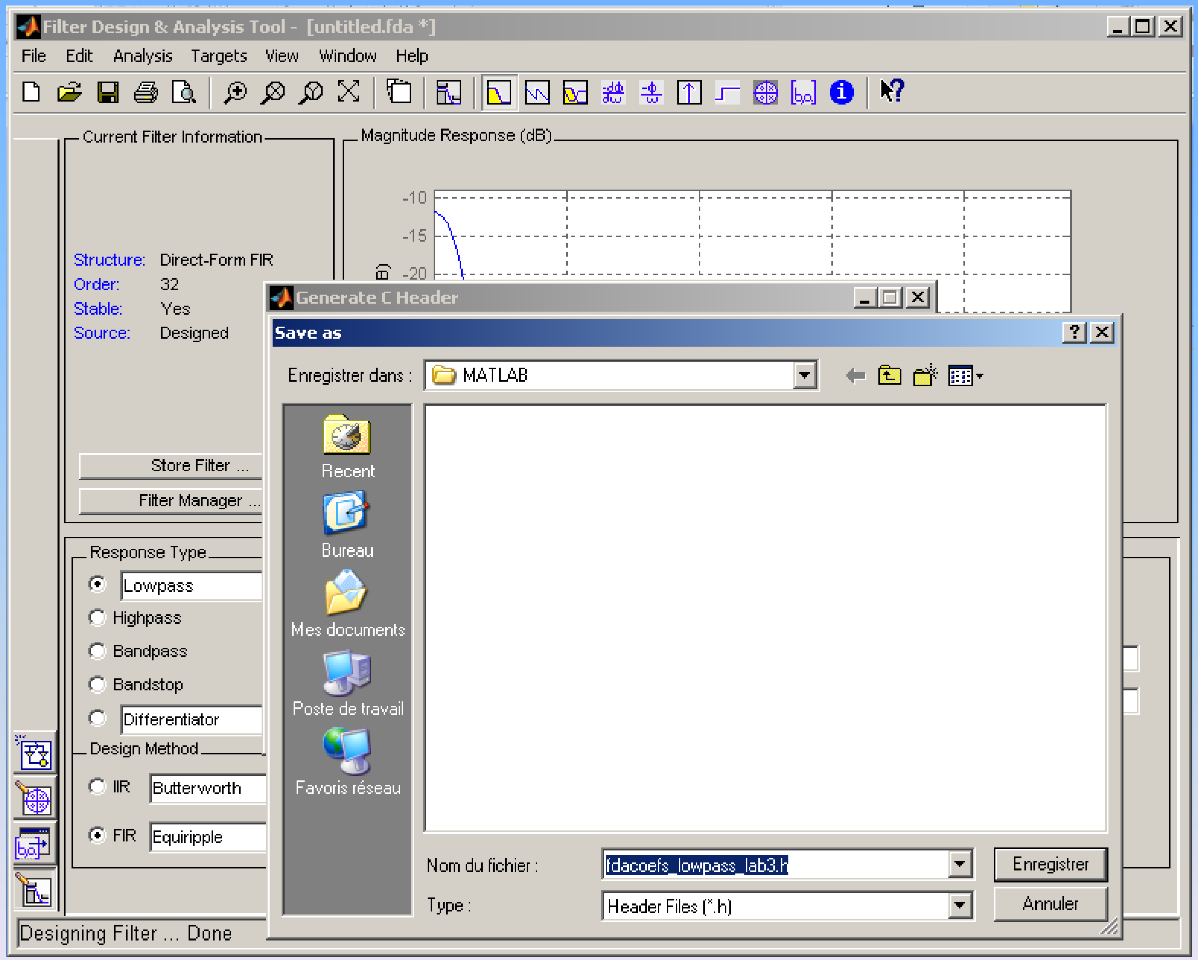
Now extract the filter coefficients



Choose 16 bit signed integers



Give a meaningful name to the file



Open the generated h-file

```
/* * Warning - Filter coefficients were
  truncated to fit specified data type.  */
const int BL = 33;
const int16_T B[33] = {423,    130,    148,
  166,    185,    203,    221,    238,
  253,    268,    281,    293,    303,
  311,    316,    320,    321,    320,
  316,    311,    303,    293,    281,
  268,    253,    238,    221,    203,
  185,    166,    148,    130,    423
};
```

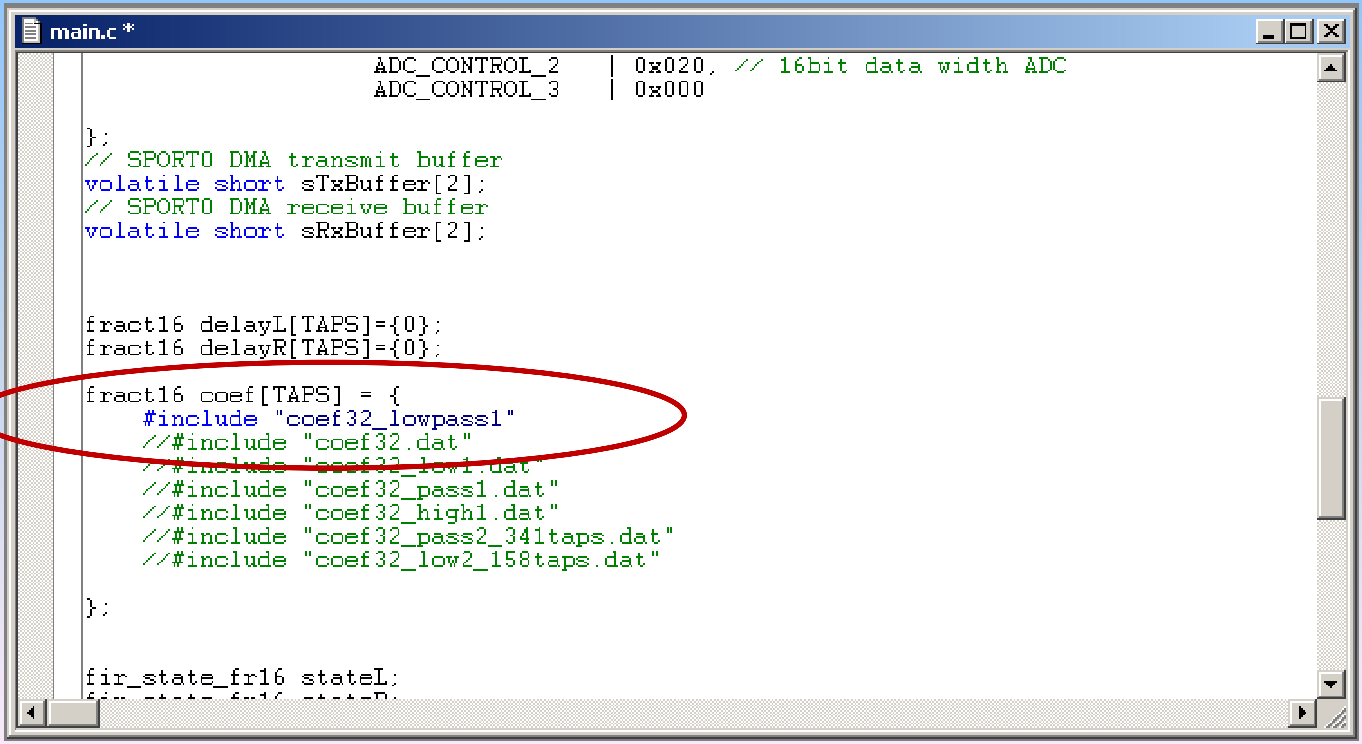
Copy ONLY the data

```
423,    130,    148,    166,    185,    203,
  221,    238,    253,    268,    281,
  293,    303,    311,    316,    320,
  321,    320,
  316,    311,    303,    293,    281,
  268,    253,    238,    221,    203,
  185,    166,    148,    130,    423
```

- paste them into a new file
- call this file for example coef32_lowpass1.dat

Add the data to the project

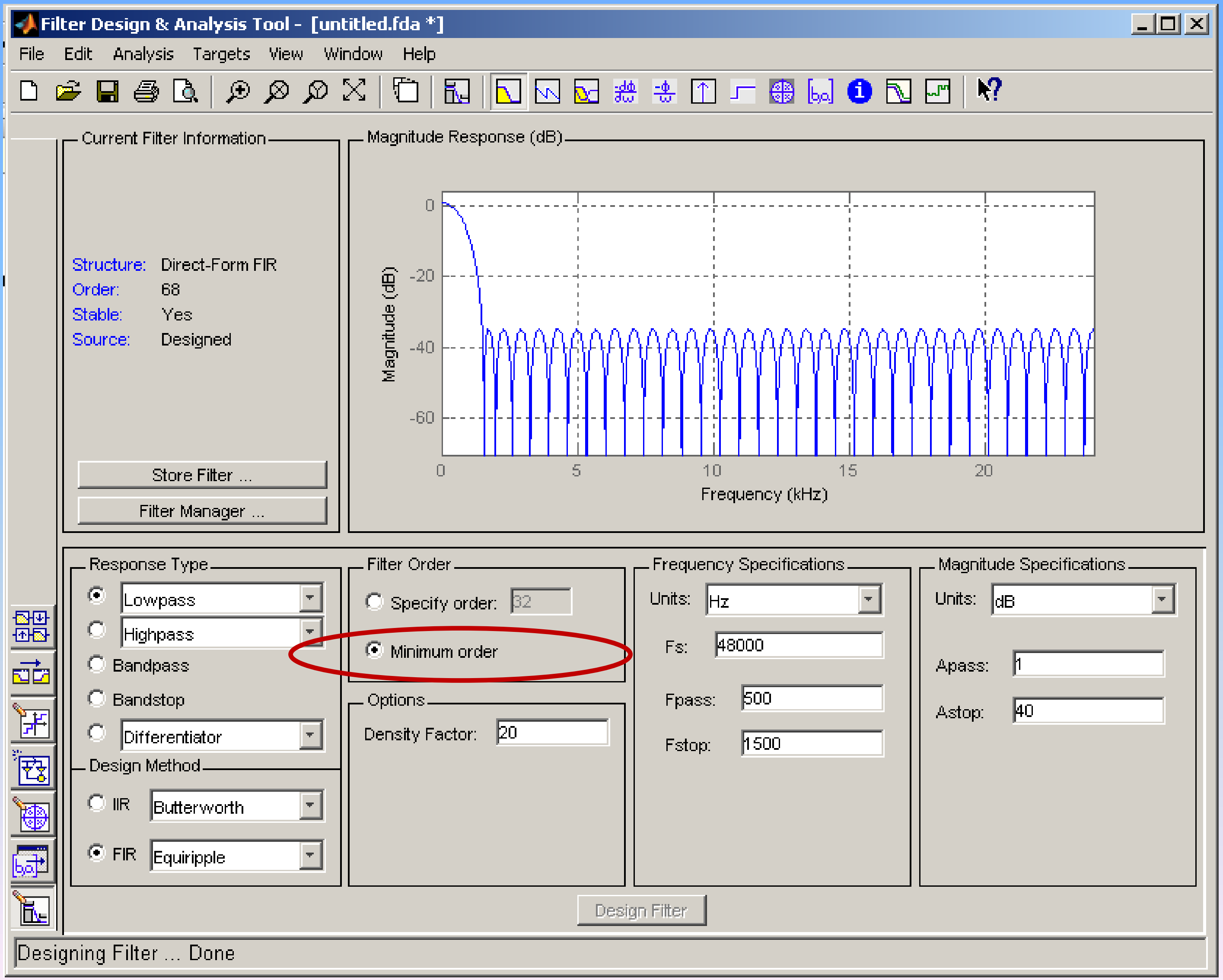
- WARNING: save the *.dat file IN THE SAME FOLDER where your Talkthrough filter VDSP++ project is



Check the result

- build the project
- run the project
- plug in a sound source
- plug in your earphones
- listen to the result
- is it low-pass filtered?
- what it the effect on the audio signal?

A better low-pass:



Check the (new) result

- build the project
- run the project
- plug in a sound source, plug in your earphones
- listen to the result
- is it low-pass filtered?
- is it better?

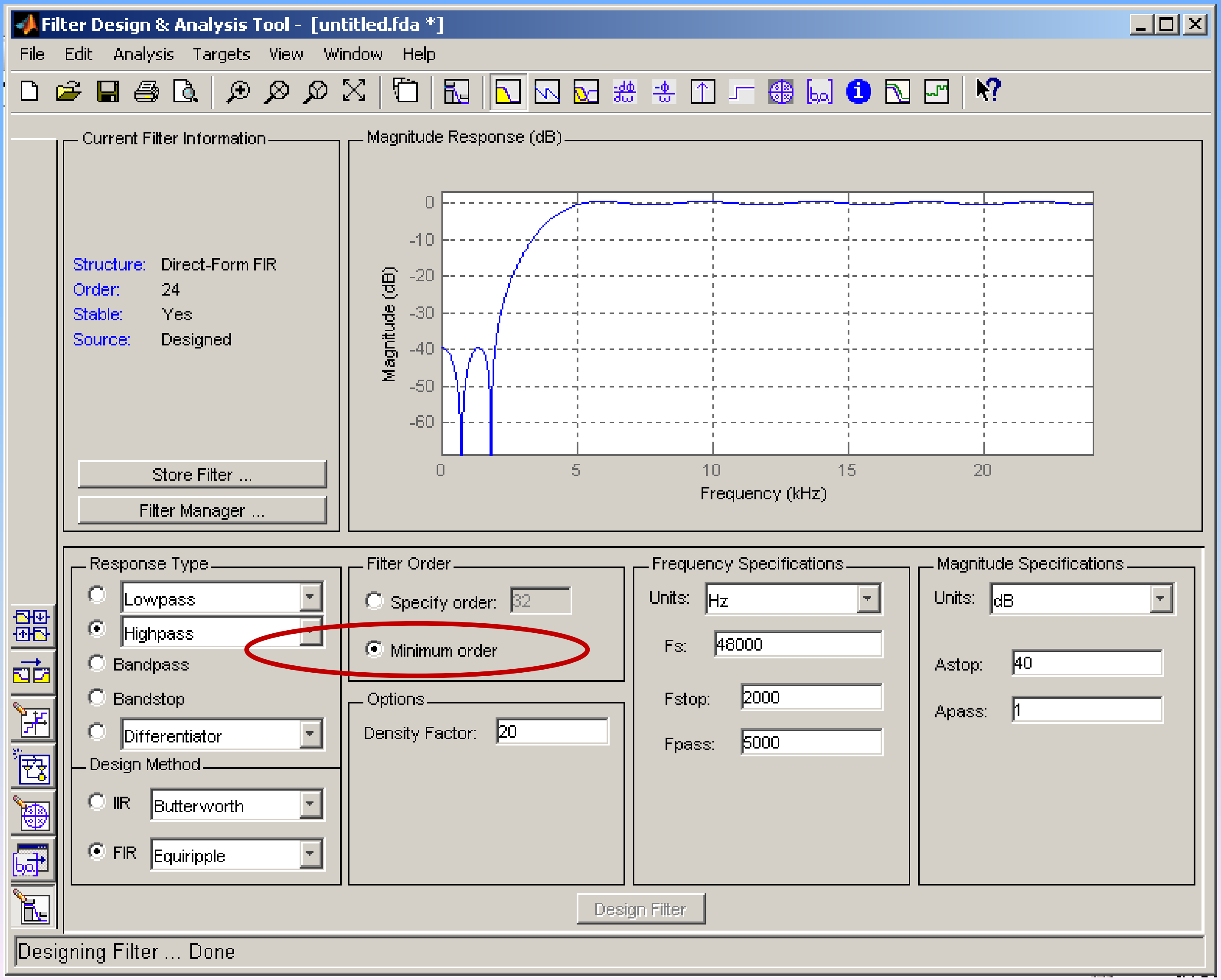
Task 2

- Design a high-pass filter:
  - FIR equiriple
  - Fstop=2000
  - Fpass=5000
  - Filter order 32
  - Wpass=1
  - Wstop=40
- Click on "design filter"

Task 2

- Design the same high-pass filter with
  - "minimum order"
- Click on "design filter"

Task 2


Filter Design & Analysis Tool - [untitled.fda *]
File Edit Analysis Targets View Window Help
Current Filter Information
Structure: Direct-Form FIR
Order: 24
Stable: Yes
Source: Designed
Store Filter ...
Filter Manager ...
Magnitude Response (dB)
Magnitude (dB)
Frequency (kHz)
Response Type
Lowpass
Highpass
Bandpass
Bandstop
Differentiator
Design Method
IIR Butterworth
FIR Equiripple
Filter Order
Specify order: 32
Minimum order
Options
Density Factor: 20
Frequency Specifications
Units: Hz
Fs: 48000
Fstop: 2000
Fpass: 5000
Magnitude Specifications
Units: dB
Astop: 40
Apass: 1
Design Filter
Designing Filter ... Done

Task 2

- implement it!
- test it!
- does it work as expected?
- explain the filter's effect on the audio signal

Task 3

- Design a band-pass filter:
  - FIR equiriple
  - Fstop1=500
  - Fpass1=1000
  - Fpass2=2000
  - Fstop2=2500
  - Filter order 32
  - Apass=1
  - Astop1,Astop2=40

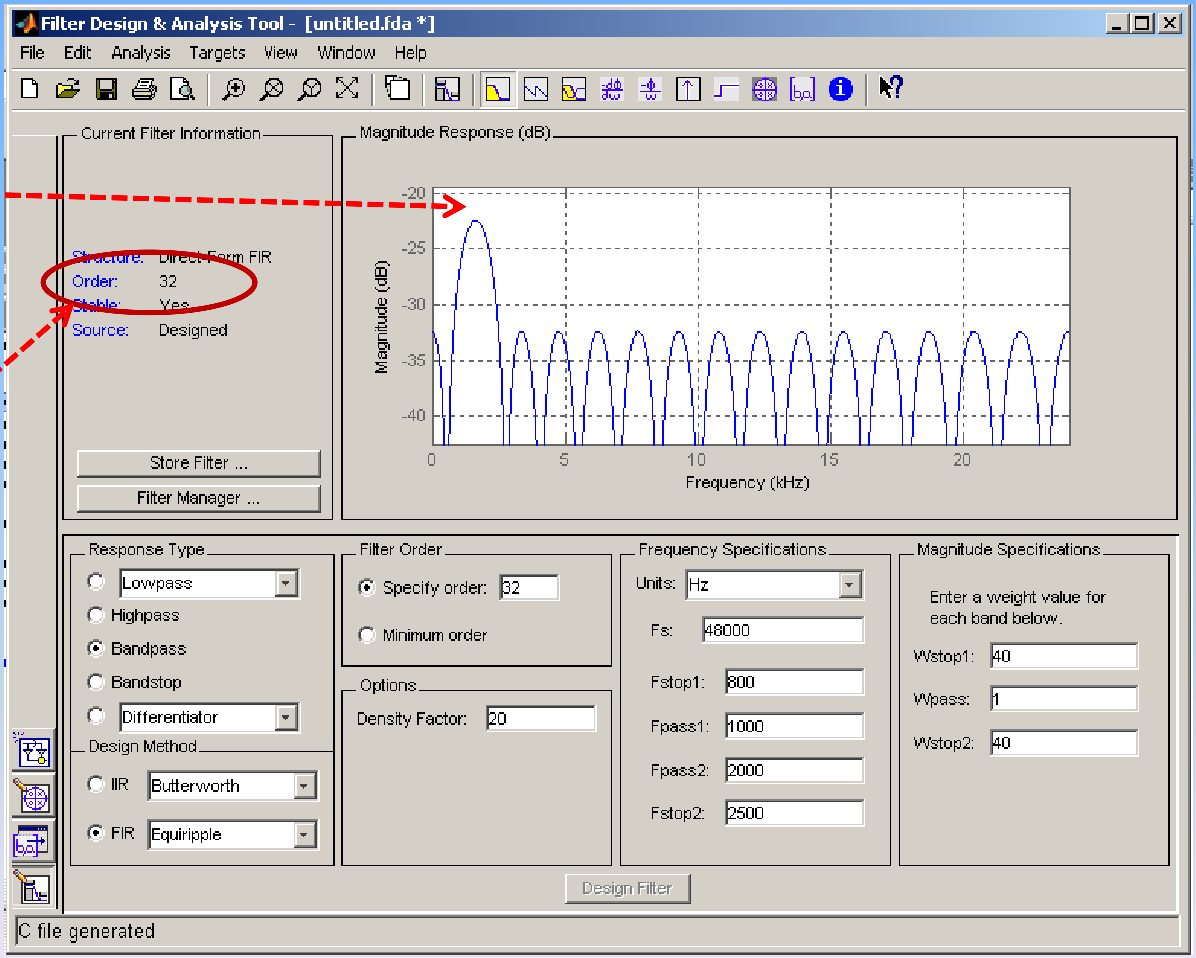
Task 3

- implement it!
- test it!
- does it work as expected? why not?

The answer is here

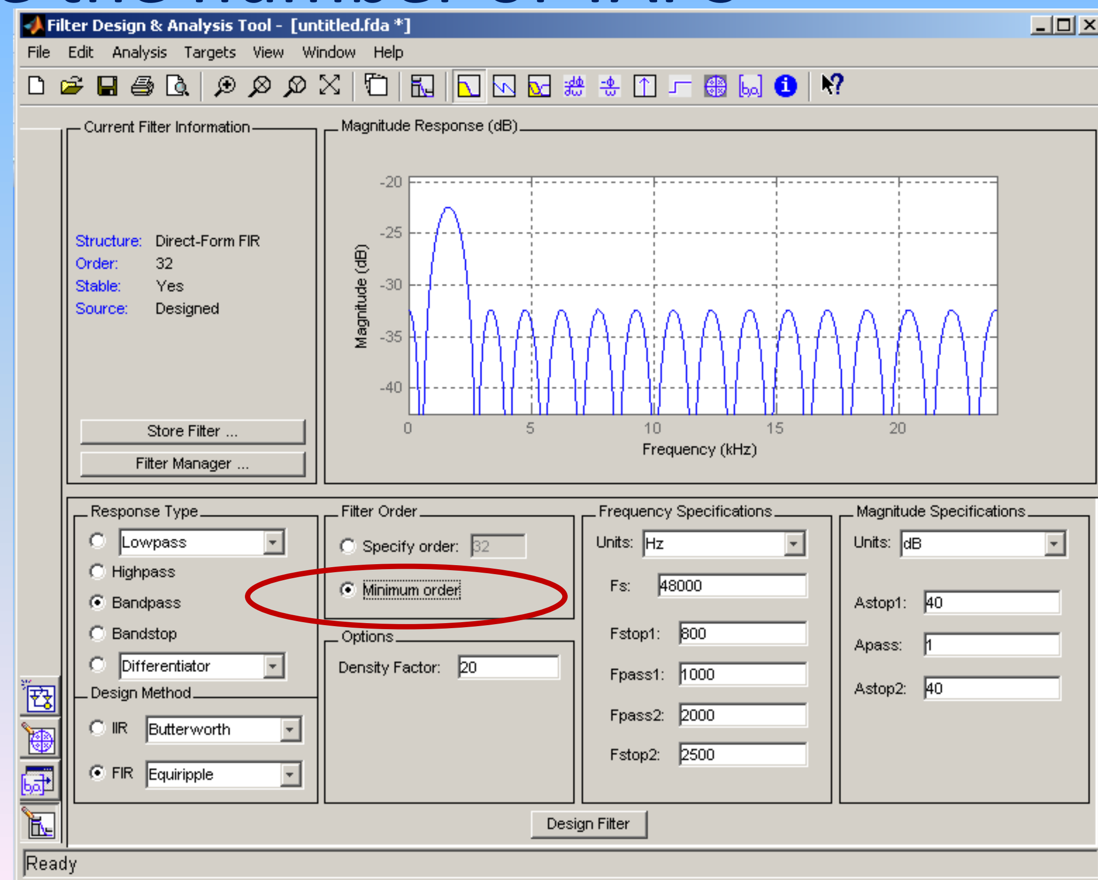
poor performance

only 32 TAPS



Solution

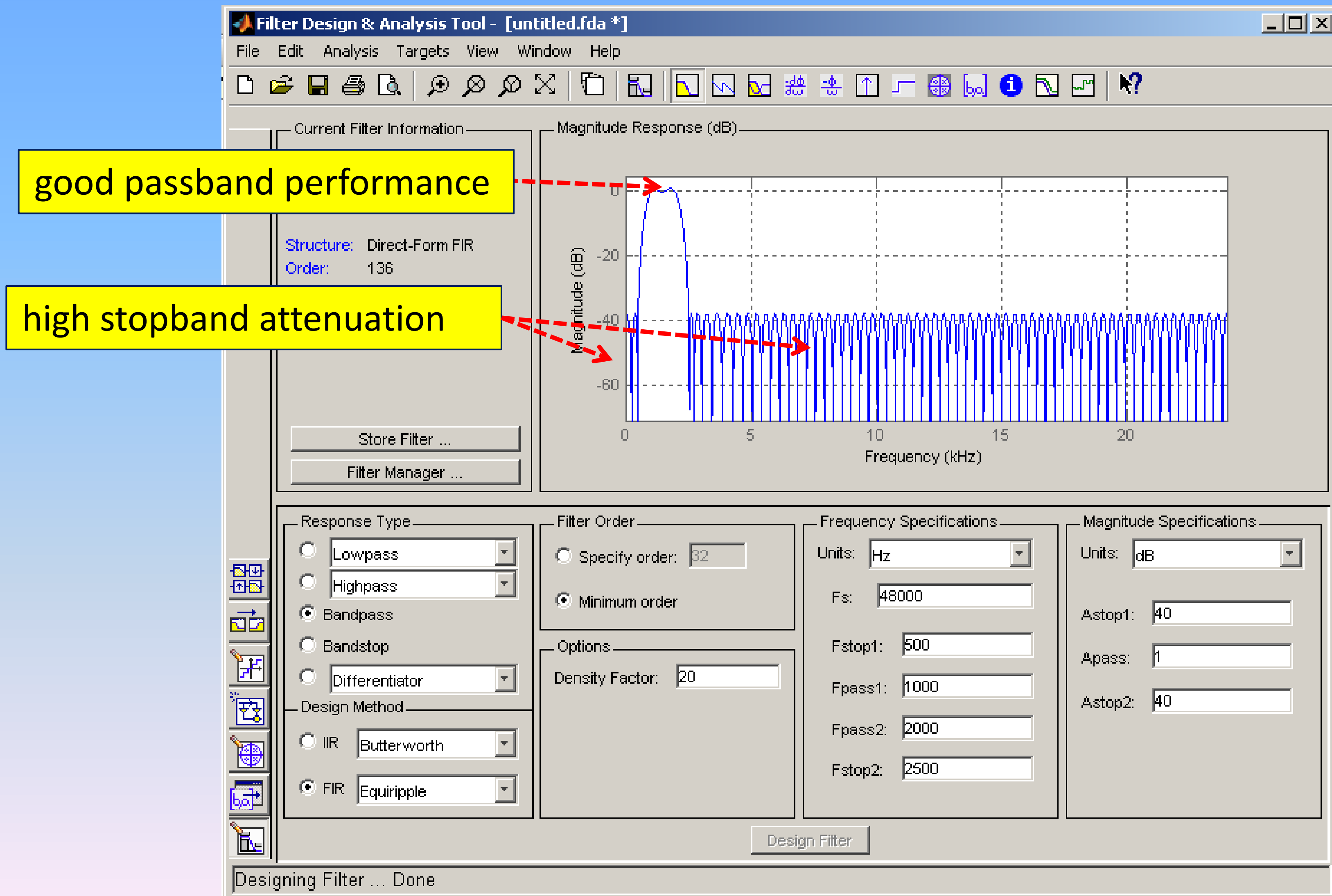
- increment the number of TAPS
- let FDAtool choose the number of TAPS



- click on "Minimum order"

- then click on "Design Filter"

Looks better



Task 3

- implement and test it!
- does it work as expected?

- warning: don't forget to change in talkthrough.h the lines:

```
// FIR number of TAPS - define it here
//#define TAPS 32
#define TAPS   136 // or whatever value
```

If you're done…

- feel free to experiment with FDAtool and implement (and test) your result on the Blackfin processor