ULBS

Universitatea “Lucian Blaga” din Sibiu

Facultatea de Inginerie Hermann Oberth
Master-Program “Embedded Systems“
Advanced Digital Signal Processing Methods
Winter Semester 2014/2015

Report 2

Building filters using specific tools

Professor: Prof. dr. ing. Mihu Ioan

Masterand: Stefan Feilmeier

14.12.2014

TABLE OF CONTENTS

1 INTRODUCTION

Everywhere in the real world we are surrounded by signals like temperature, sound, voltage and so on. All those signals are analogue, i.e. they are **continuous** in the time domain. To extract useful information, to further process or analyse them, it is necessary to use filters. Those filters can be implemented as “analogue[1] filters” or as “digital filters”, as shown in Fig. 1.

“**Analogue filters**”, like operational amplifiers, are able to directly process the analogue signal very fast, without any loss in information due to conversions. Unfortunately their implementation can be complicated and their flexibility is restricted.

“**Digital filters**” allow an easier implementation and enable a much wider range of possibilities and features – at the cost of converting the signal from analogue to digital (“Analogue to Digital Converter” – ADC) and vice versa (“Digital to Analogue Converter” – DAC).

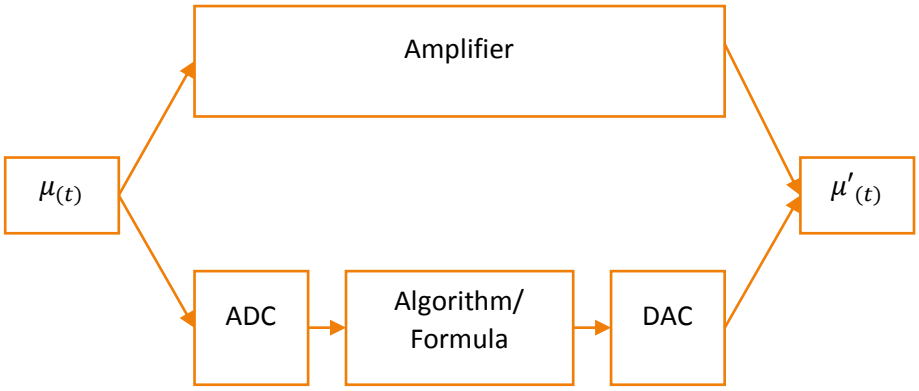

Fig. 1 Analogue and Digital Filters

The ADC is sampling and quantizing the continuous analogue signal (Fig. 2), returning a digital signal in the form of a **stream of numbers**. The minimal sampling frequency is defined by the following “Shannon Theorem”:

$$sampling\ frequeny \geq 2 \times highest\ frequency\ in\ spectrum$$

For example for music, a value of $\geq 40\ kHz$ is recommended, to avoid aliasing and damaging of the input signal. The maximum sampling frequency is up to the system designer and only limited by the amount of data that may be generated. The number of quantization steps is also up to the system designer. In order to optimally use the digital computing data structure, the distinct quantization values are generally defined according to the number of bits that are used per value, e. g. 8 bits allow $2^8 = 256$ distinct values, 16 bits allow for $2^{16} = 65536$ values and so on. The rule is, that per increase of 1 bit, the rate of noise will decrease by 6 dB.

Then an algorithm or formula, that uses the last N samples of a series to calculate the new value, is applied to the digital stream of numbers, before in the DAC it is converted back to an analogue signal using interpolation.

[1] Throughout this paper the British English spelling “analogue“ is used in favour of American English “analog“

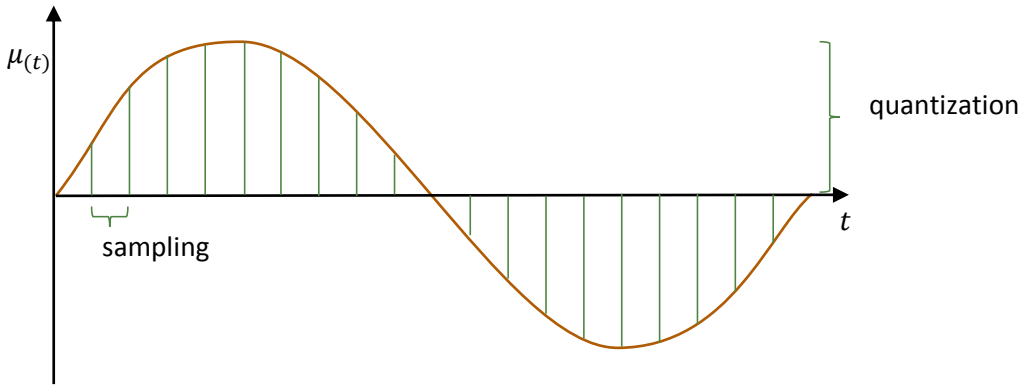

Fig. 2 Analogue to Digital Converter: Sampling and Quantization

Signals are a set of each other overlapping sine waves with different frequencies. To visualize these frequencies of a signal, it is necessary to transform the signal from **time domain** to **frequency domain** using the algorithms “**Fourier Theorem**” (for periodic signals) or “**(Fast) Fourier Transform**” (for non-periodic signals).

Fig. 3 shows two signals in both time and frequency domain (spectrum). The signal in the first row consists of three simple sine waves with frequencies 1000, 2000 and 3000 Hz. The second example is an audio signal taken from a WAV-file.

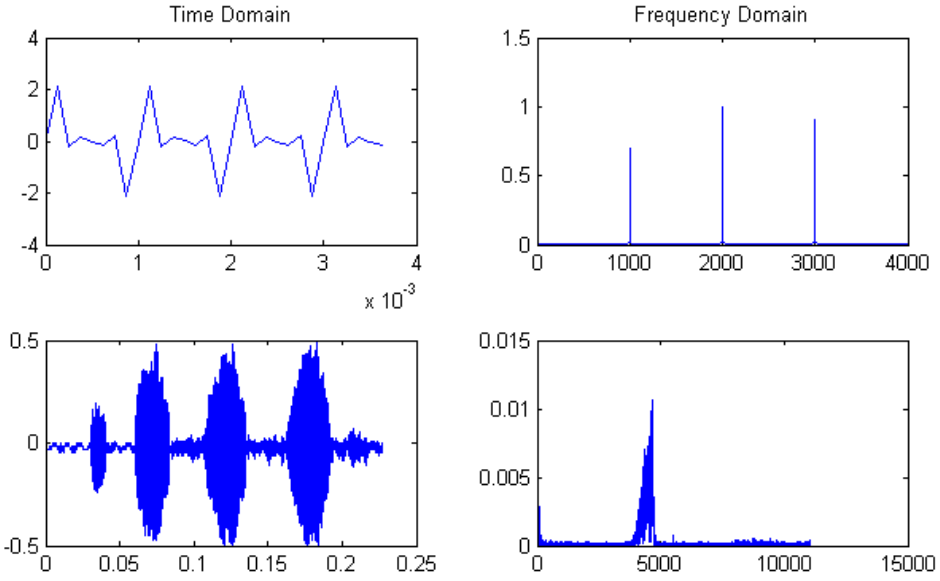

Fig. 3 Example signals in time and frequency domain

The purpose of this paper is to show the process of building simple digital filters. The general approach is to:

1. <u>Define requirements for frequency characteristics.</u>
   This paper concentrates on two simple filters in frequency domain: “Low Pass Filter” (LPF) and “Band Pass Filter”.
2. <u>Find the appropriate algorithm.</u>
   This paper mentions non-recursive “Finite Impulse Response” (FIR) and recursive “Infinite

Impulse Response (IIR)” filters. In the examples FIR filters are used and ways to identify the needed coefficients for the calculation are shown.

3. Implement the filter.
   The resulting algorithm is applied to the two example signals.

2 BUILD FILTER USING TOOLS

This paper focuses on the creation of non-recursive “Finite Impulse Response” (FIR) filters. They are applied using the following formula:

$$y_{[n]} = \sum_{k=0}^{N-1} h_{[k]} \times x_{[n-k]}$$

An alternative to FIR filters are recursive “Infinite Impulse Response (IIR)” filters. They are recursive, because they reuse their outputs as inputs. The general formula is the following:

$$y_{[n]} = \sum_{k=0}^{N-1} h_{[k]} \times x_{[n-k]} - \sum_{j=1}^{M} h_{[k]} \times y_{[n-j]}$$

The first step in implementing a filter is to calculate the coefficients for the chosen algorithm. Instead of doing the required calculations manually, this paper relies on well-established algorithms in Matlab and other tools.

All filters in this paper are defined as being normalized between 0 and 1 Hz. To apply them on real problems, the need to be converted using the “rule of three”.

Some algorithms require the resulting number of coefficients (N) to be given as a parameter. To compare the efficiency of each of them, each function is tested with different parameters:

```
N1 = 10;
N2 = 25;
N3 = 40;
```

Each function is tested to provide the coefficients to approximate the “Low Pass Filter (LPF)” and “Band Pass Filter (BPF)” shown in Fig. 4:

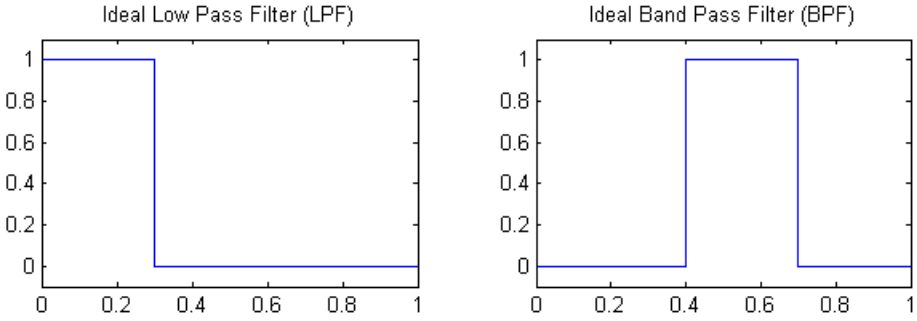

Fig. 4 Ideal Low Pass Filter (LPF) and Band Pass Filter (BPF)

Low Pass Filter:

```
f1 = [0 0.3 0.3 1];
a1 = [1 1   0   0];
```

Band Pass Filter:

```
f2 = [0 0.4 0.4 0.7 0.7 1];
a2 = [0 0   1   1   0   0];
```

As result of each algorithm, an array of coefficients like the following is expected:

```
b11 =
  Columns 1 through 6
   -0.0052   -0.0080    0.0134    0.1057    0.2405    0.3072
  Columns 7 through 11
    0.2405    0.1057    0.0134   -0.0080   -0.0052
```

2.1 MATLAB: WINDOW-BASED FINITE IMPULSE RESPONSE FILTER DESIGN (FIR1)

“fir1 implements the classical method of windowed linear-phase FIR digital filter design [1]. It designs filters in standard lowpass, highpass, bandpass, and bandstop configurations. By default the filter is normalized so that the magnitude response of the filter at the center frequency of the passband is 0 dB.” [Matlab]

Low Pass Filter:

```
ff1 = 0.3;
b11 = fir1(N1, ff1, 'low');
b21 = fir1(N2, ff1, 'low');
b31 = fir1(N3, ff1, 'low');
```

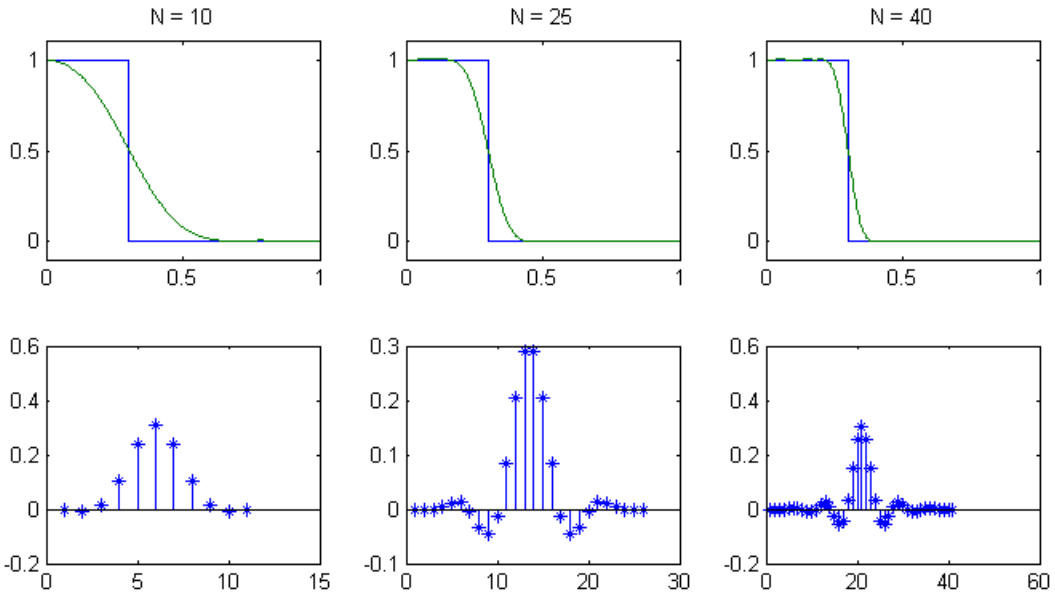

Fig. 5 Frequency characteristics and coefficients for LPF using fir1

Band Pass Filter:

```
ff2 = [0.4, 0.7];
b12 = fir1(N1, ff2, 'band');
b22 = fir1(N2, ff2, 'band');
b32 = fir1(N3, ff2, 'band');
```

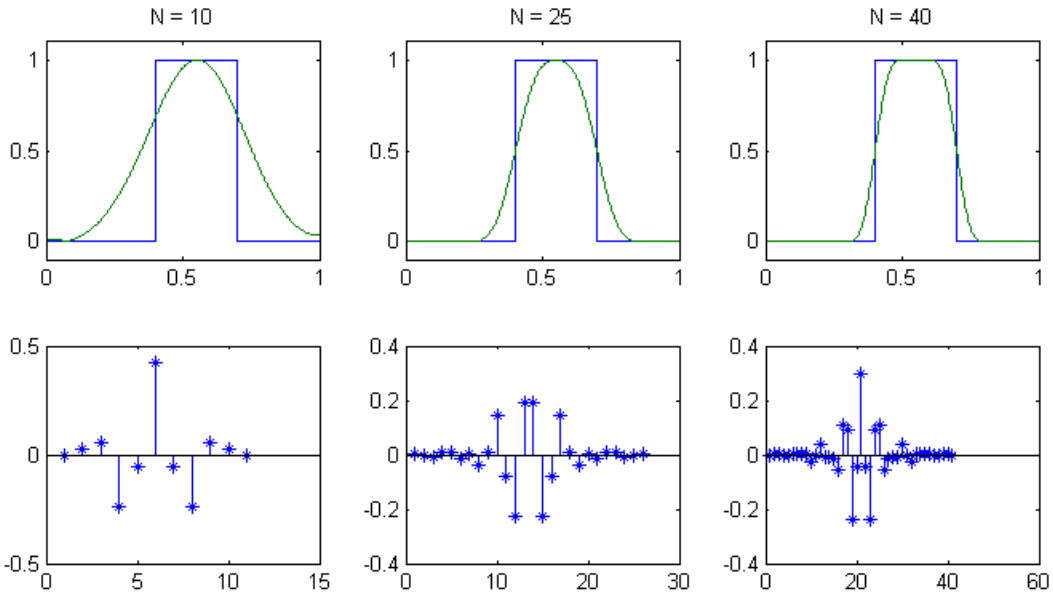

Fig. 6 Frequency characteristics and coefficients for BPF using fir1

2.2 MATLAB: FREQUENCY SAMPLING-BASED FINITE IMPULSE RESPONSE FILTER DESIGN (FIR2)

> *fir2 designs frequency sampling-based digital FIR filters with arbitrarily shaped frequency response. [Matlab]*

“fir2” can be used for individual filter designs that differ from the standard filters (LPF, HPF, BPF, BSF). Nevertheless, it is mentioned here to show the general functionality. The result also shows, that there can be significant differences in the resulting filter’s coefficients.

Low Pass Filter:

```
b11 = fir2(N1, f1, a1);
b21 = fir2(N2, f1, a1);
b31 = fir2(N3, f1, a1);
```

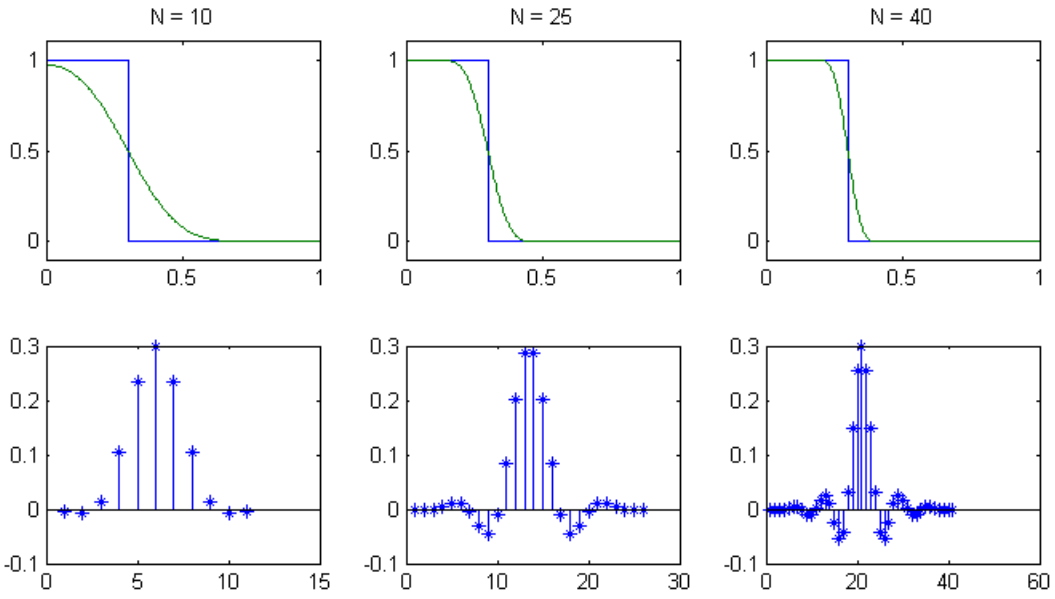

Fig. 7 Frequency characteristics and coefficients for LPF using fir2

Band Pass Filter:

```
b12 = fir2(N1, f2, a2);
b22 = fir2(N2, f2, a2);
b32 = fir2(N3, f2, a2);
```

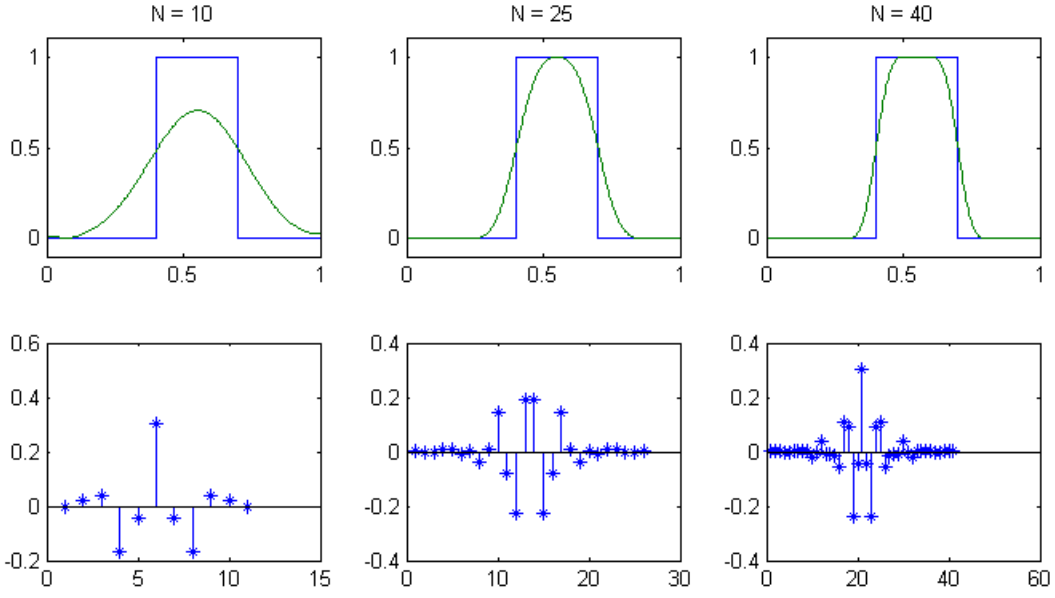

Fig. 8 Frequency characteristics and coefficients for BPF using fir2

2.3 MATLAB: FILTER DESIGN & ANALYSIS TOOL (FDATOOL)

Using Matlab's "Filter Design & Analysis Tool", a filter can be designed in a graphical way, using a wide variety of algorithms. The following examples show settings for LPF and BPF filters, trying to reassemble the previous behaviour.

Low Pass Filter:

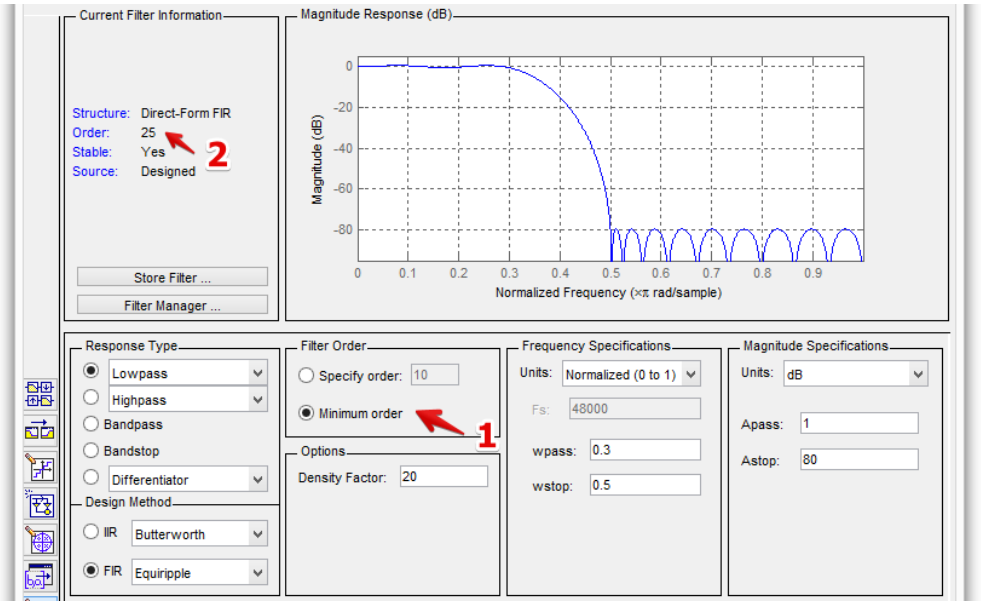

Fig. 9 Filter Design & Analysis Tool for LPF

The "Minimum order" setting in (1) leaves the selection of the order (N) to the algorithm, according to the "Frequency-" and "Magnitude specifications". The algorithm concludes to use an order of $N = 25$.

Band Pass Filter:

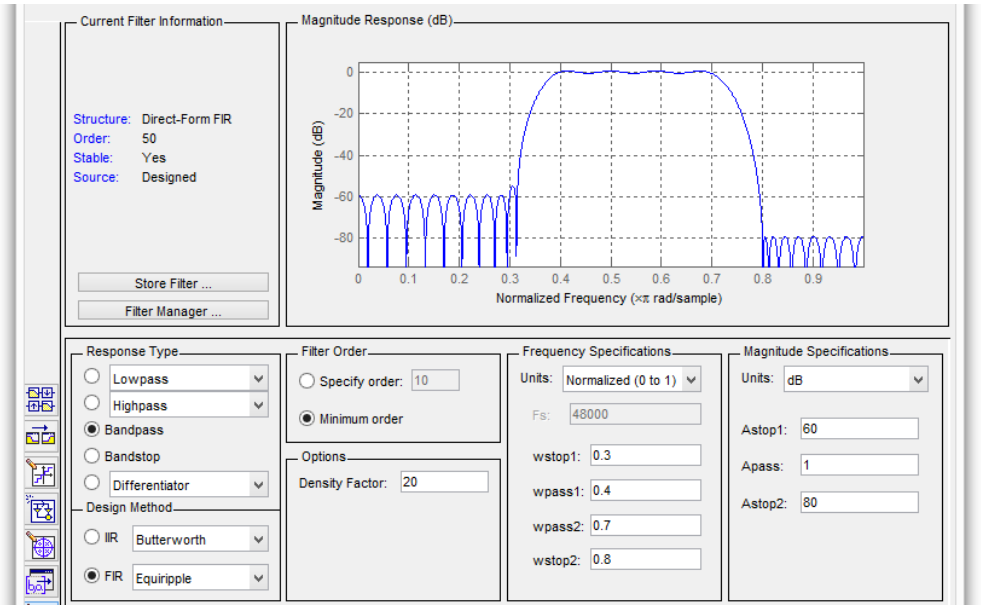

Fig. 10 Filter Design & Analysis Tool for BPF

2.4 TFILTER

Searching the Internet for “create fir filter”, one can find many tools, dedicated to the creation of FIR filters. One of those tools is “TFilter”[2]. It is a free online tool that allows, similarly to Matlab’s Filter Design & Analysis Tool, an easy definition and instantaneous graphical representation of filters. The resulting coefficients may be directly exported to C code; either as floating point or fixed point numbers.

Low Pass Filter:

Fig. 11 TFilter for LPF

Band Pass Filter:

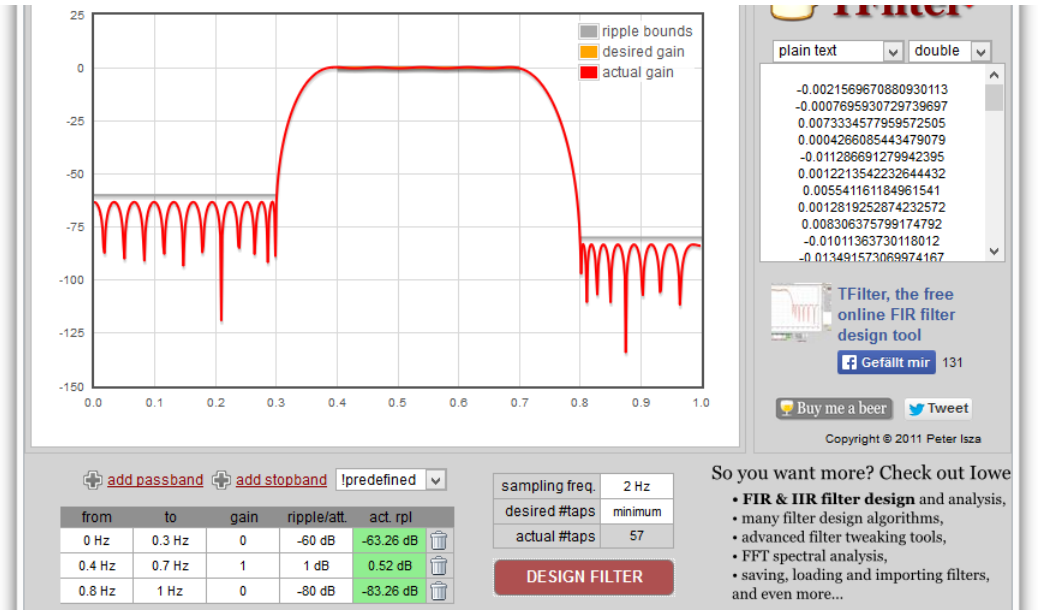

Fig. 12 TFilter for BPF

[2] TFilter: http://t-filter.appspot.com/fir/index.html

3 IMPLEMENT FILTER

All the previously shown methods result in an array of coefficients which sufficiently describe a filter. For an example implementation, the “Low Pass Filter” built with Matlab’s “Filter Design & Analysis Tool” is used and applied to both example signals. They are therefor exported into the workspace, using fdatool’s export utility:

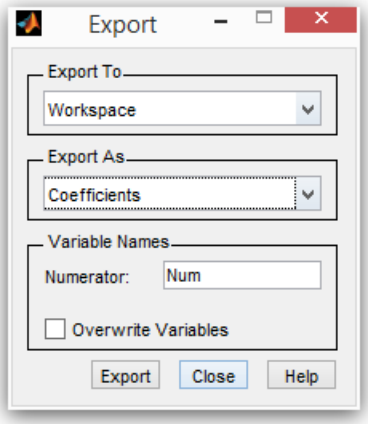

Fig. 13 Export filter from fdatool

Alternatively, the array could be created manually from the calculated coefficients and used together with the input signal “x” as parameter for the filter command:

```
Num = [-0.0017, -0.0074, -0.0155, -0.0163, -0.0005, 0.0251, ...];
y = filter(Num, 1, x);
```

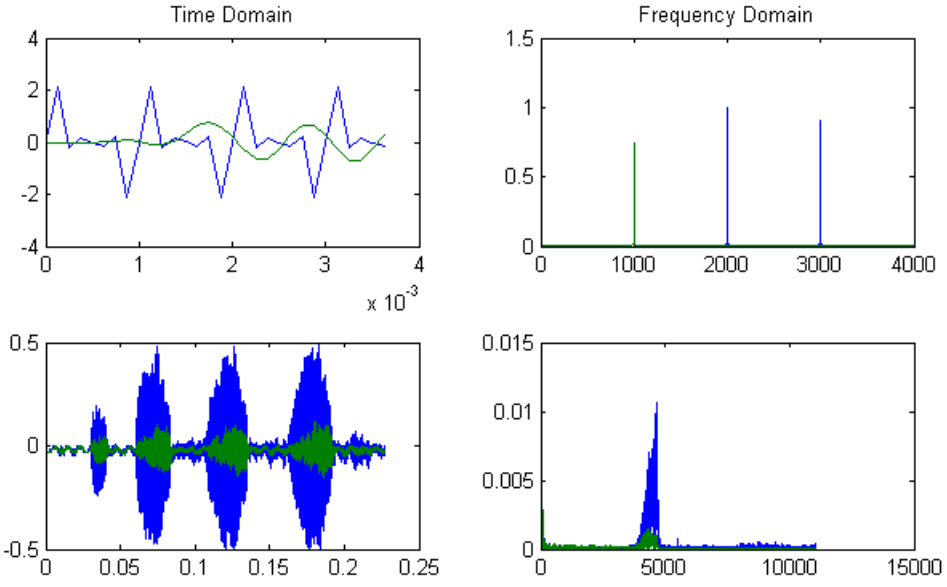

Fig. 14 Example signals before (blue) and after (green) applying a LPF

After applying the “Low Pass Filter”, it can be clearly seen, that the result is as expected and high frequencies are removed from the resulting signal spectrum.

TABLE OF FIGURES