UM1625
User manual

Biquad library software expansion for STM32Cube

Introduction

This user manual describes the software interface and requirements of the Biquad (BIQ) module. It describes how to integrate the module into a main program like the Audio STM32Cube expansion software and it provides a rough understanding of the underlying algorithm.

The BIQ library is part of X-CUBE-AUDIO firmware package.

Contents

List of tables

List of figures

1 Module overview

1.1 Algorithm function

The BIQ module is a cascade of Infinite Impulse Response second order filters. The coefficients of each second order section (SOS) can be configured independently. The overall BIQ transfer function can thus be computed for many purposes: standard filter design such as low-pass, high-pass, notch, peak, high-shelf and low-shelf or arbitrary frequency response for transducer equalization.

The current implementation is using a 32-bit resolution for the coefficient format, which allows extreme filter design (for example low-pass filter close to 0 Hz, high-pass filter close to Nyquist frequency).

1.2 Module configuration

The BIQ module supports Mono and Stereo interleaved 16 or 32 bit I/O data, with a maximum input frame size of 480 stereo samples. This limitation corresponds to a 10 ms scheduling at a 48 kHz sampling frequency. The maximum number of the second order section, which can be cascaded, is set to 10, which is a trade-off between the equalization possibility and the computing load.

Several versions of the module are available depending on the I/O format, the Cortex Core and the used tool chain:

- BIQ_CM4_IAR.a / BIQ_CM4_GCC.a / BIQ_CM4_Keil.lib: library with 16 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set
- BIQ_32b_CM4_IAR.a / BIQ_32b_CM4_GCC.a / BIQ_32b_CM4_Keil.lib: library with 32 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set
- BIQ_CM7_IAR.a / BIQ_CM7_GCC.a / BIQ_CM7_Keil.lib: library with 16 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set
- BIQ_32b_CM7_IAR.a / BIQ_32b_CM7_GCC.a / BIQ_32b_CM7_Keil.lib: library with 32 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M7instruction set

1.3 Resource summary

Table 1 contains the module requirements for the CPU, Flash, Stack and RAM memories and frequency (MHz). The core frequency (MHz) value is an average measured on real hardware.

Scratch RAM is the memory that can be shared with other process running on the same priority level. Between two calls to BIQ routines, this scratch RAM can thus be re-used by other processes.

Table 1. Resource summary

I/O	Core	BIQ Configuration	Flash code ..text (Byte)	Flash data .rodata (Byte)	Stack (Byte)	Persistent RAM (Byte)	Scratch RAM (Byte)	Frequency (MHz)
16bit	M4	1 SOS - Mono 48kHz - 16bits I/O	2446	8	70	600	4	0.69
		10 SOS - Stereo 48kHz - 16bits I/O						13.7
	M7	1 SOS - Mono 48kHz - 16bits I/O	2396	8	70	600	4	0.48
		10 SOS - Stereo 48kHz - 16bits I/O						9.6
32bit	M4	1 SOS - Mono 48kHz - 32bits I/O	2510	8	70	600	4	0.73
		10 SOS - Stereo 48kHz - 32bits I/O						14.6
	M7	1 SOS - Mono 48kHz - 32bits I/O	2456	8	70	600	4	0.6
		10 SOS - Stereo 48kHz - 32bits I/O						12

2 Module Interfaces

Two files are needed to integrate the BIQ module: BIQ_xxx_CMy_zzz.a/.lib library and *biquad_df1_cascade_glo.h* header file. They contain all definitions and structures to be exported to the application SW.

Note: The audio_fw_glo.h file is a generic header file common to all audio modules; it must be included in the audio framework.

2.1 APIs

Six generic functions have a software interface to the main program:

- biquad_df1_cascade_reset
- biquad_df1_cascade_setParam
- biquad_df1_cascade_getParam
- biquad_df1_cascade_setConfig
- biquad_df1_cascade_getConfig
- biquad_df1_cascade_process

2.1.1 biquad_df1_cascade_reset function

This procedure initializes the static memory of the BIQ module and initializes static parameters with default values.

```
int32_t biquad_df1_cascade_reset(void *persistent_mem_ptr, void
*scratch_mem_ptr);
```

Table 2. biquad_df1_cascade_reset

I/O	Name	Type	Description
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Input	scratch_mem_ptr	void *	Pointer to internal scratch memory
Returned value	-	int32_t	Error value

This routine must be called at least once at initialization time, when the real time processing has not started.

2.1.2 biquad_df1_cascade_setParam function

This procedure writes module static parameters from the main framework to the module's internal memory. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameter with values which cannot be changed during the module processing.

```
int32_t biquad_df1_cascade_setParam(biquad_df1_cascade_static_param_t
*input_static_param_ptr, void *persistent_mem_ptr);
```

Table 3. biquad_df1_cascade_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	biquad_df1_cascade_static_param_t*	Pointer to static parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value	-	int32_t	Error value

2.1.3 biquad_df1_cascade_getParam function

This procedure gets the module static parameters from the module internal memory to the main framework. It can be called after the reset routine and before the start of the real time processing. It handles the static parameters, i.e. the parameters with values which cannot be changed during the module processing.

```
int32_t biquad_df1_cascade_getParam(biquad_df1_cascade _static_param_t
*input_static_param_ptr, void *persistent_mem_ptr);
```

Table 4. biquad_df1_cascade_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	biquad_df1_cascade_static_param_t *	Pointer to static parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value		int32_t	Error value

2.1.4 biquad_df1_cascade_setConfig function

This procedure sets the module dynamic parameters from the main framework to the module internal memory. It can be called at any time during processing.

```
int32_t biquad_df1_cascade_setConfig( biquad_df1_cascade_dynamic_param_t
*input_dynamic_param_ptr, void *persistent_mem_ptr);
```

Table 5. biquad_df1_cascade_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	biquad_df1_cascade_dynamic_param_t *	Pointer to dynamic parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Returned value		int32_t	Error value

2.1.5 biquad_df1_cascade_getConfig function

This procedure gets module dynamic parameters from the internal persistent memory to the main framework. It can be called at any time during processing.

```
int32_t biquad_df1_cascade_getConfig(biquad_df1_cascade_dynamic_param_t
*input_dynamic_param_ptr, void *persistent_mem_ptr);
```

Table 6. biquad_df1_cascade_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	biquad_df1_cascade_dynamic_param_t *	Pointer to dynamic parameters structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Output		int32_t	Error value

2.1.6 biquad_df1_cascade_process function

This procedure is the module's main processing routine. It should be called at any time, to process each frame.

```
int32_t biquad_df1_cascade_process(buffer_t *input_buffer, buffer_t
*output_buffer, void *persistent_mem_ptr);
```

Table 7. biquad_df1_cascade_process

I/O	Name	Type	Description
Input	input_buffer	buffer_t *	Pointer to input buffer structure
Output	output_buffer	buffer_t *	Pointer to output buffer structure
Input	persistent_mem_ptr	void *	Pointer to internal persistent memory
Output		int32_t	Error value

This process routine can run in place, meaning that the same buffer can be used for input and output at the same time.

2.2 External definitions and types

2.2.1 Input and output buffers

The BIQ library is using extended I/O buffers which contain, in addition to the samples, some useful information on the stream such as the number of channels, the number of bytes per sample and the interleaving mode.

An I/O buffer structure type, as described below, must be followed; otherwise an error will be returned:

```
typedef struct {
    int32_t    nb_channels;
    int32_t    nb_bytes_per_Sample;
    void       *data_ptr;
    int32_t    buffer_size;
    int32_t    mode;
} buffer_t;
```

Table 8. Input and output buffers

Name	Type	Description
nb_channels	int32_t	Number of channels in data: 1 for mono, 2 for stereo
nb_bytes_per_Sample	int32_t	16-bit = 2, 32-bit = 4
data_ptr	void *	Pointer to data buffer (must be allocated by the main framework)
buffer_size	int32_t	Number of samples per channel in the data buffer
mode	int32_t	In case of stereo stream, left and right channels can be interleaved. 0 = not interleaved, 1 = interleaved.

2.2.2 Returned error values

Table 9 contains the possible returned error values:

Table 9. Returned error values

Definition	Value	Description
BIQ_ERROR_NONE	0	No error
BIQ_UNSUPPORTED_INTERLEAVING_MODE	-1	Input data is stereo/not interleaved
BIQ_UNSUPPORTED_NB_CHANNELS	-2	Input data is neither mono nor stereo
BIQ_UNSUPPORTED_NB_OF_BYTEPERSAMPLE	-3	Input data is not a 16 or 32 bit sample format

2.3 Static parameters structure

The Biquad coefficients are set using the corresponding static parameter structure before calling the biquad_df1_cascade_setParam() function.

```
struct biquad_df1_cascade_static_param {
    uint8_t  nb_sos;
    int32_t  biquad_coeff[6*MAX_NB_SOS];
}
```

Table 10. Static parameters structure

Name	Type	Description
nb_sos	uint8_t	Number of second order sections for the current configuration. Valid range is [1:10]
biquad_coeff[]	int32_t	Array of SOS coefficients and post-shift values. It contains 6 parameters for each SOS. See *Table 11: biquad_coeff structure*. For each SOS, coefficient values are computed with the following formula: $coeff_{int32} = round(coeff_{dec} \times 2^{31 - post\_shift})$

Table 11. biquad_coeff structure

SOS coefficient and post-shift values	SOS
post_shift	1^{st} SOS
b0	
b1	
b2	
a1	
a2	
post_shift	2^{nd} SOS
b0	
...	
a2	
post_shift	Last SOS
b0	
...	
a2	

2.4 Dynamic parameters structure

It is possible to change the Biquad configuration by setting the new coefficients in the dynamic parameter structure before calling the `biquad_df1_cascade_setConfig()` function. An “enable” parameter is available to enable or disable the Biquad module.

```
struct biquad_df1_cascade_static_param {
    uint8_t   enable;
    uint8_t   nb_sos;
    int32_t   biquad_coeff[6*MAX_NB_SOS];
}
```

Table 12. Dynamic parameters structure

Name	Type	Description
enable	uint8_t	0: no filtering is done, the output buffer is equal to the input buffer 1: the input buffer is processed and stored in the output buffer
nb_sos	uint8_t	Number of second order sections for the current configuration. Valid range is [1:10]
biquad_coeff[]	int32_t	Array of SOS coefficients and post-shift values. It contains 6 parameters for each SOS. See *Table 11: biquad_coeff structure*. For each SOS, coefficients value are computed with the following formula: $\mathrm{coeff}_{\mathrm{int32}} = \mathrm{round}(\mathrm{coeff}_{\mathrm{dec}} \times 2^{31 - \mathrm{post\_shift}})$

3 Algorithm description

3.1 Processing steps

The BIQ module is based on a biquadratic implementation of an infinite impulse response (IIR) with two zeros and two poles. The second order section (SOS) equation is as follows:

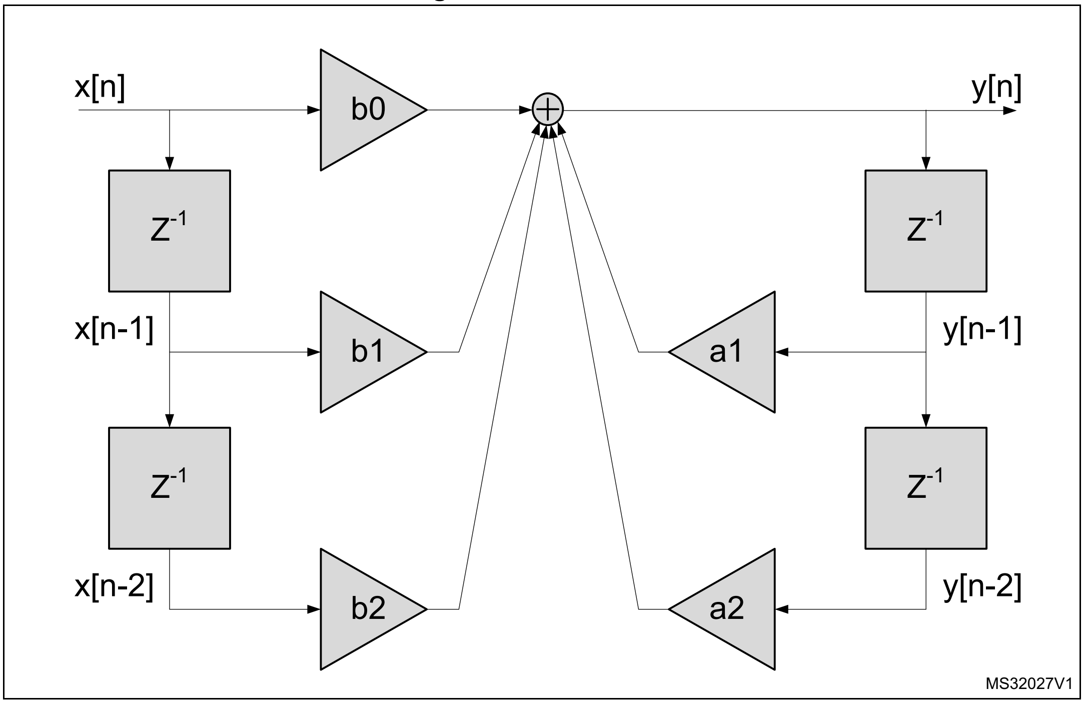
$$a0 \times y(n) = b0 \times x(n) + b1 \times x(n-1) + b2 \times x(n-2) - a1 \times y(n-1) - a2 \times y(n-2)$$

All coefficients are normalized to a0, so that the final equation has 5 coefficients:

$$y(n) = \frac{b0}{a0} \times x(n) + \frac{b1}{a0} \times x(n-1) + \frac{b2}{a0} \times x(n-2) - \frac{a1}{a0} \times y(n-1) - \frac{a2}{a0} \times y(n-2)$$

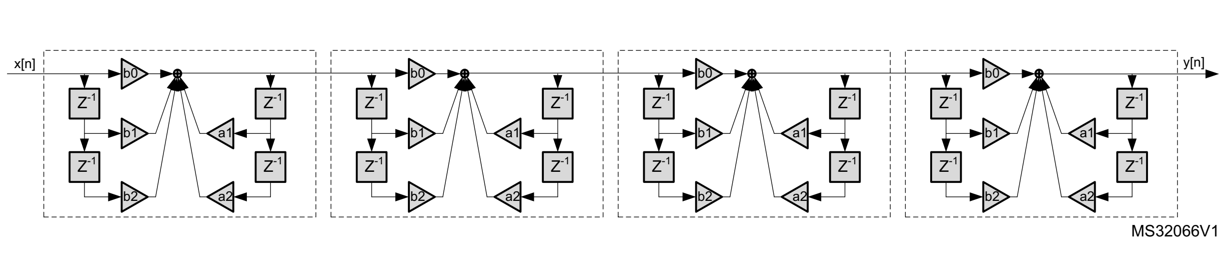
The direct-form I is the most straightforward code implementation and has the advantage of being stable. The 'numerator' part is first computed, which lowers the risk of saturating and the state variables can easily be bounded. *a1* and *a2* signs are reverted so that the equation becomes a combination of multiple/accumulate operations.

Figure 1. BIQ module



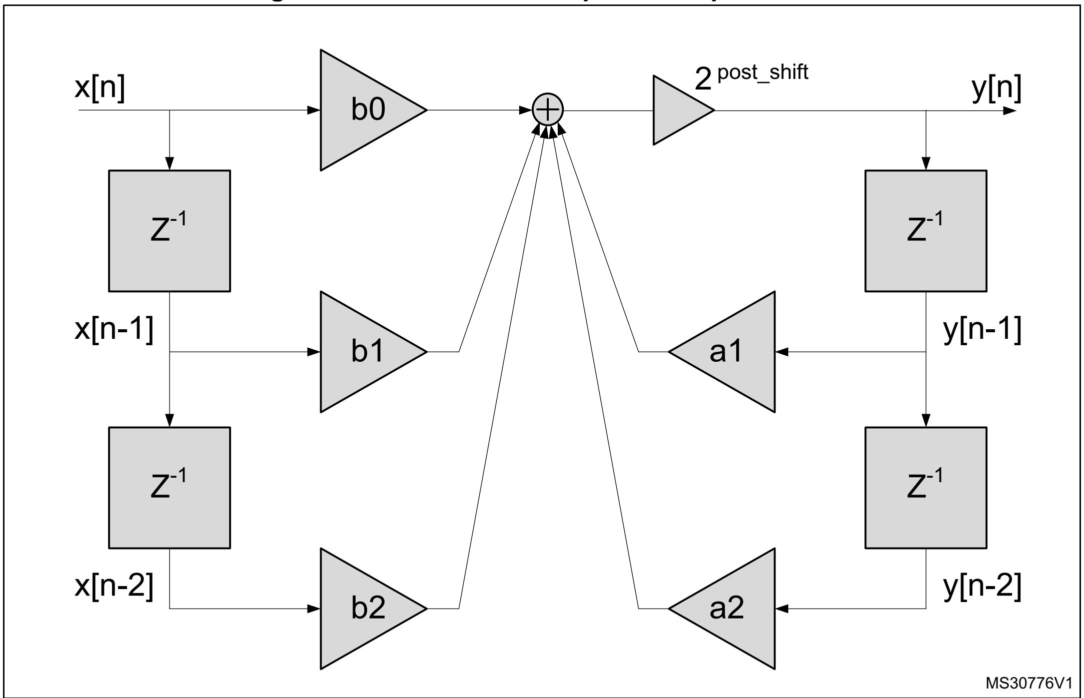
For filters with an order greater than 2, it is possible to cascade up to 10 SOS. The result of the previous SOS is used as the next SOS input:

Figure 2. Cascaded BIQ modules



Coefficients are represented as fractional values, restricted to [-1.0:1.0]. To allow coefficients to exceed this range, a post-shift parameter is added in the SOS structure:

Figure 3. BIQ module with post-shift parameter



The overflow is managed by saturating the output value to +/- 1.0. The input signal is scaled down so that it prevents any overflow in case of a filter design with a gain < 1.0. In case of a filter with a gain > 1.0, more scaling must be done on the filter coefficients or the input signal, otherwise clipping will occur.

The processing loop can be of any size, with respect to the maximum number of samples which is 960. But one should notice that the implementation is optimized with loop-unrolling and is optimal for loop processing sizes which are multiples of 8.

3.2 Data formats

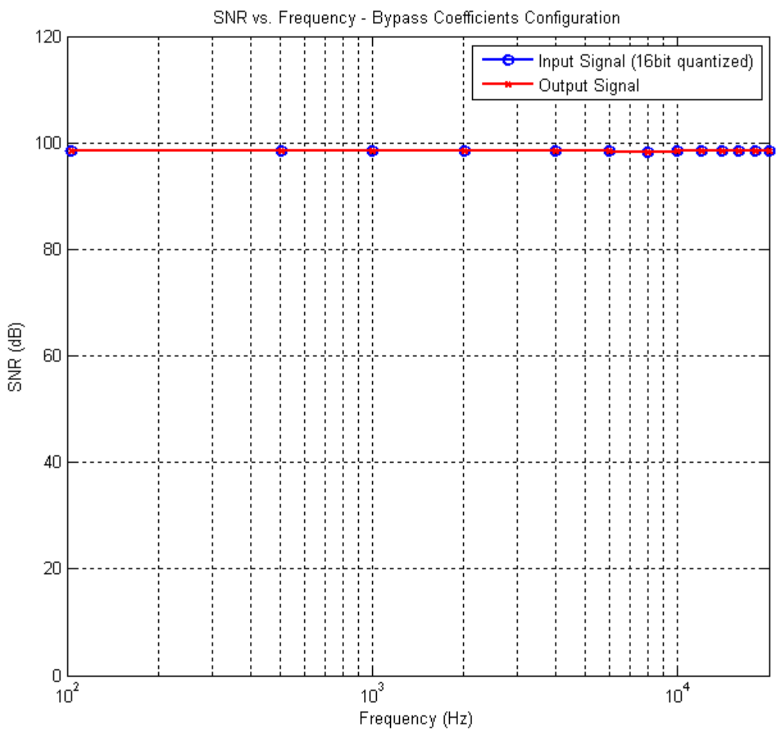
The input of a BIQ module is expected to be an audio stream, mono or stereo/interleaved, in a 16 or 32 bit format. All operations are done with a 32-bit resolution.

3.3 Performances measurements

3.3.1 SNR

The BIQ module is configured with bypass coefficients, and feed with full-scale sinusoid at different frequencies. The SNR measurement is then performed at different frequencies and compare to the input signal. Below is the plot of input/output SNR comparison:

Figure 4. SNR



One can see that SNR is not altered at all by the BIQ processing.

The SNR formula used is:

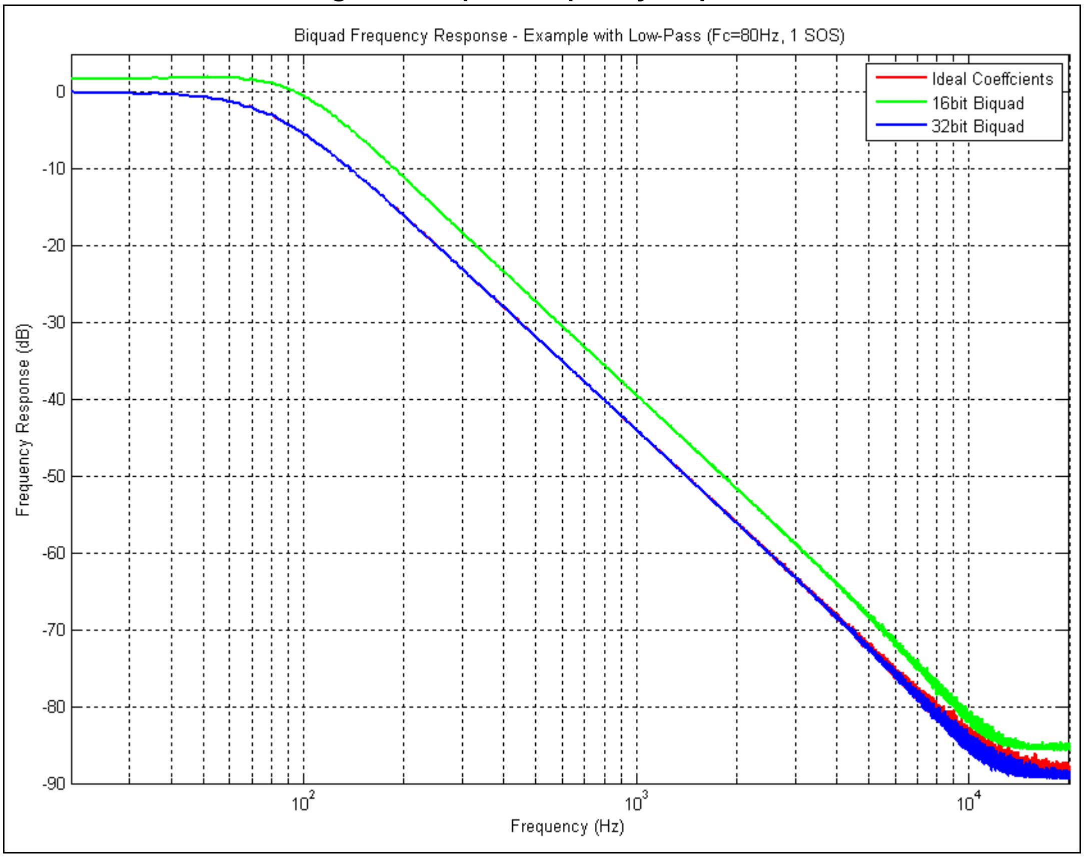
$$SNR = 10 \times \log 10\left(\frac{SignalEnergy}{NoiseEnergy}\right)$$

3.3.2 Frequency response measurements

It is proposed to compare the frequency response measured in the ideal case with Matlab and with the BIQ module. The design which is done is a low-pass filter, a 2nd order slope, a sampling rate of 48 kHz and a cut-off frequency of 80 Hz.

The ratio cut-off/half-sampling-rate is equal to 0.0033. Below is the plot where 3 frequency responses are compared: Matlab filtering function (double floating point format), 16-bit Biquad (16-bit fixed point format) filter and BIQ module (32-bit fixed point format). One can see that a 16-bit resolution is not enough for such low cut-off frequency, and that the Matlab filtering function and BIQ module are comparable.

Figure 5. Biquad frequency response



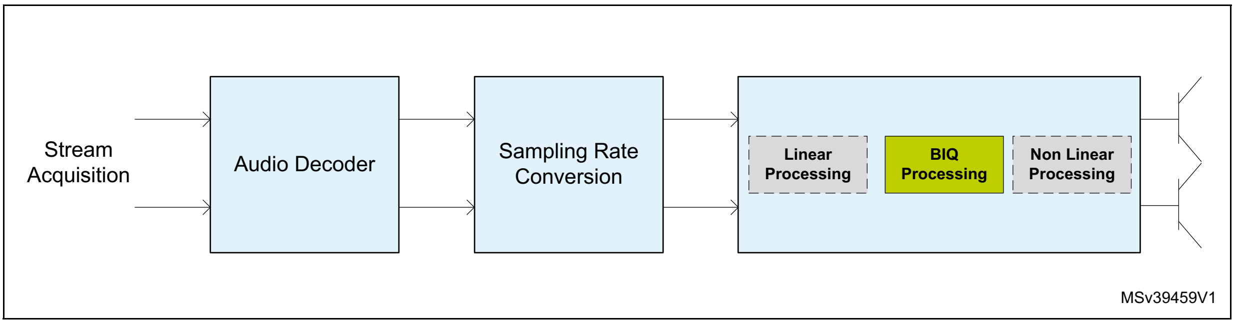
4 Application description

4.1 Recommendations for optimal setup

The BIQ module can be executed at any place in an audio processing chain. Nevertheless, for a usage such as a transducer equalizer, it is recommended to call this module close to the equalized transducer, i.e. near the DAC for a speaker and close to the ADC for a microphone.

A speaker or a microphone equalization can be needed when the transducer frequency response shows a lot of variations in its magnitude. Flattening a speaker frequency response can bring more fidelity in the music reproduction and thus can be found as “better” than a non-equalized one. Before using the Biquad Design Tool to compute such an equalizer, it is needed to measure the speaker frequency response with an appropriate acoustic equipment. *Figure 6* shows the optimal setup

Figure 6. Basic audio chain

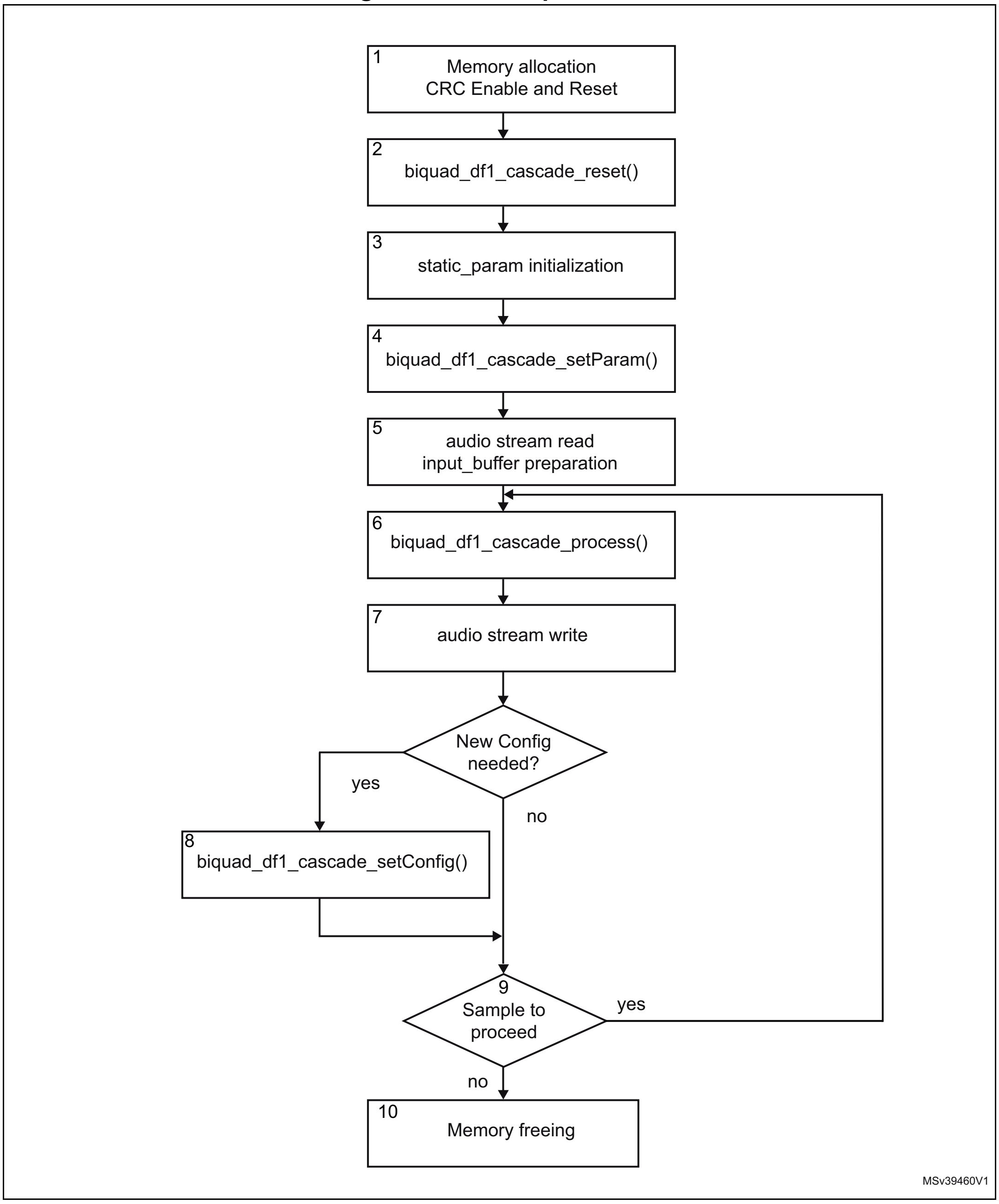


4.1.1 Module integration example

Cube expansion BIQ integration examples are provided on STM32F746G-Discovery and STM32F469I-Discovery boards. Refer to the provided integration code for more details.

4.1.2 Module API calls

Figure 7. API call procedure



1. As explained above, Biquad persistent and scratch structures have to be allocated, as well as the input and output buffer, according to the structures defined in the *Introduction*.
2. Once the memory is allocated, the call to biquad_df1_cascade_reset() function will initialize the internal variables and set a bypass filter as the default configuration.
3. The Biquad configuration for the desired filter response can now be set by initializing the static_param structure.
4. A call to the biquad_df1_cascade_setParam() function will then configure the Biquad internal memory according to the desired configuration.
5. The audio stream is read from the proper interface, and the input_buffer structure has to be filled according to the stream characteristics (number of channels, sample rate, interleaving and data pointer). The output buffer structure has to be set as well.
6. A call to the process will execute the filtering part of the Biquad library and produce the processed input buffer in the output buffer.
7. The output audio stream can now be written in the proper interface.
8. If needed, the user can set new dynamic parameters and call the biquad_df1_cascade_setConfig() function to update the module configuration.
9. If the application is still running and has new input samples to process, the processing loop goes back to step 5, otherwise is over.
10. Once the processing loop is over, allocated memory has to be freed.

5 How to tune and run the application

The STM32 audio framework provides an example of application for the BIQ library.

The filter used in the integration example is of low-pass type, with a cut-off frequency of 150Hz.

For any filter design, user should refer to the Biquad Design Tool, described in the following chapter.

6 Biquad design tool

Many tools are available for the Biquad filter design. However, in order to avoid the end user to make mistakes on the specific Biquad coefficient format interface, the library is delivered with the “Biquad Design Tool”. This tool proposes a pre-defined and arbitrary filter design, with a computation of the associated set of parameters ready-to-use with the Biquad library. This section details the installation of the tool and is a guideline on how to make a design.

6.1 Tool installation

Biquad design tool is developed with Matlab. Then compiled with the Matlab Compiler, it can be run on any PC equipped with Windows and Matlab Component Run-time (MCR), with no need for a license.

MCR must have been installed before executing the setup file for the tool. Run the `MCRInstaller.exe` file and follow the instructions:

Figure 8. Install the Matlab compiler runtime

Figure 9. Accept the licence agreement

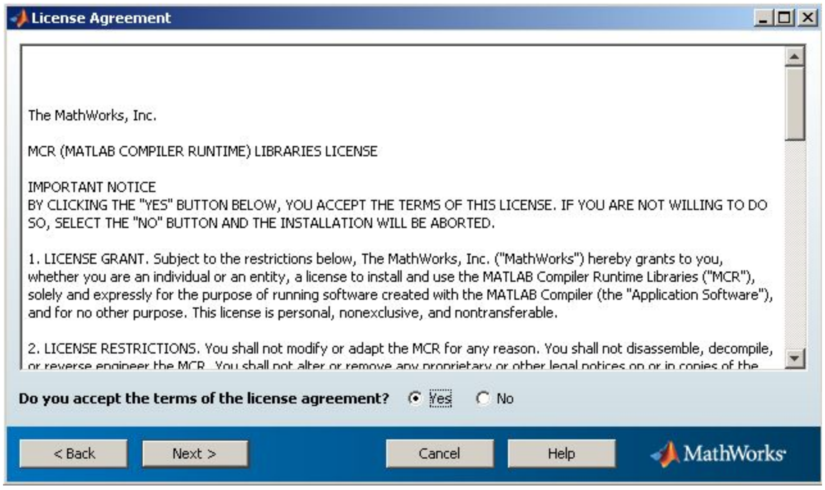


Figure 10. File extraction

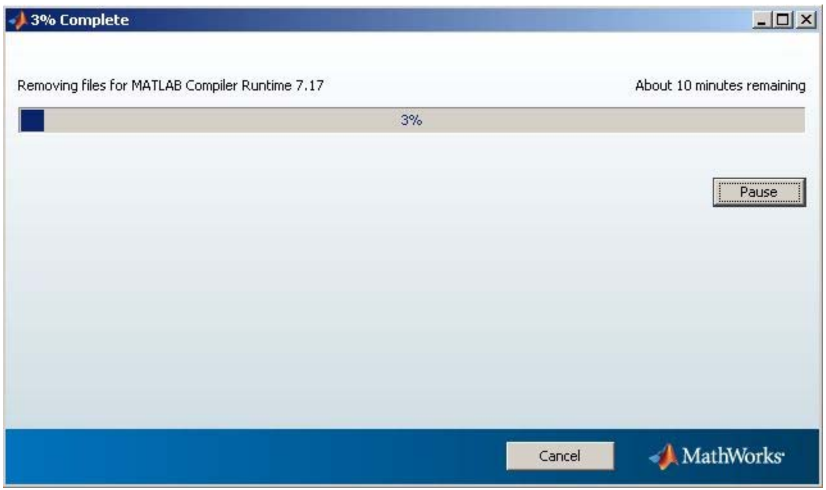
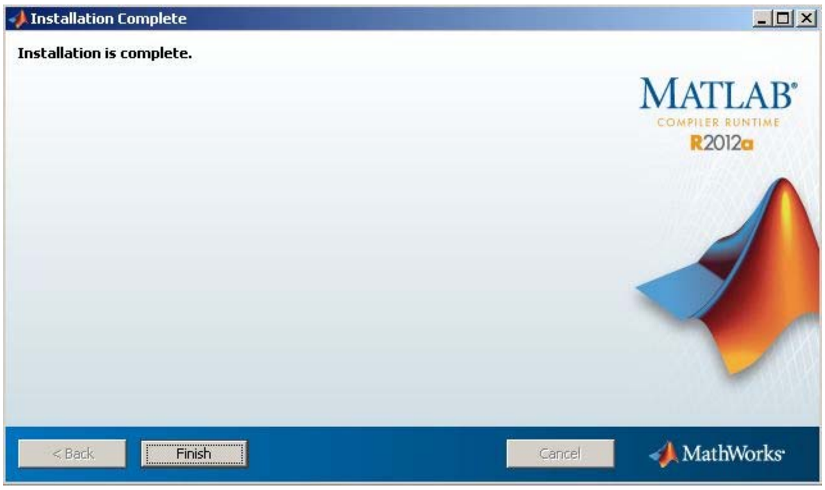


Figure 11. Installation complete



The MCR is a run-time component that should be installed once. Even if the Biquad Design Tool is updated later, there will be no need to update the MCR.

One can now run the `biquad_filter_design.exe` file and follow the instructions:

Figure 12. Instructions: select the file location

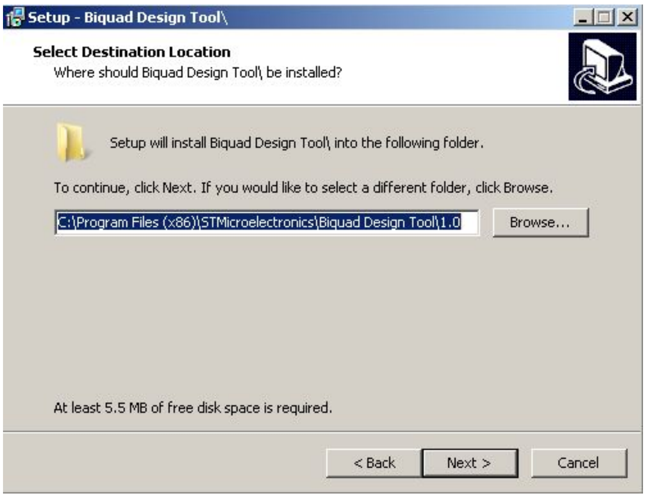
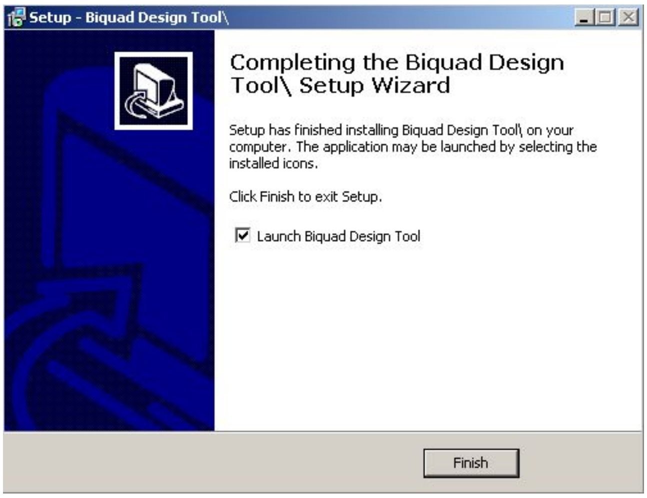


Figure 13. Instructions: launch the application

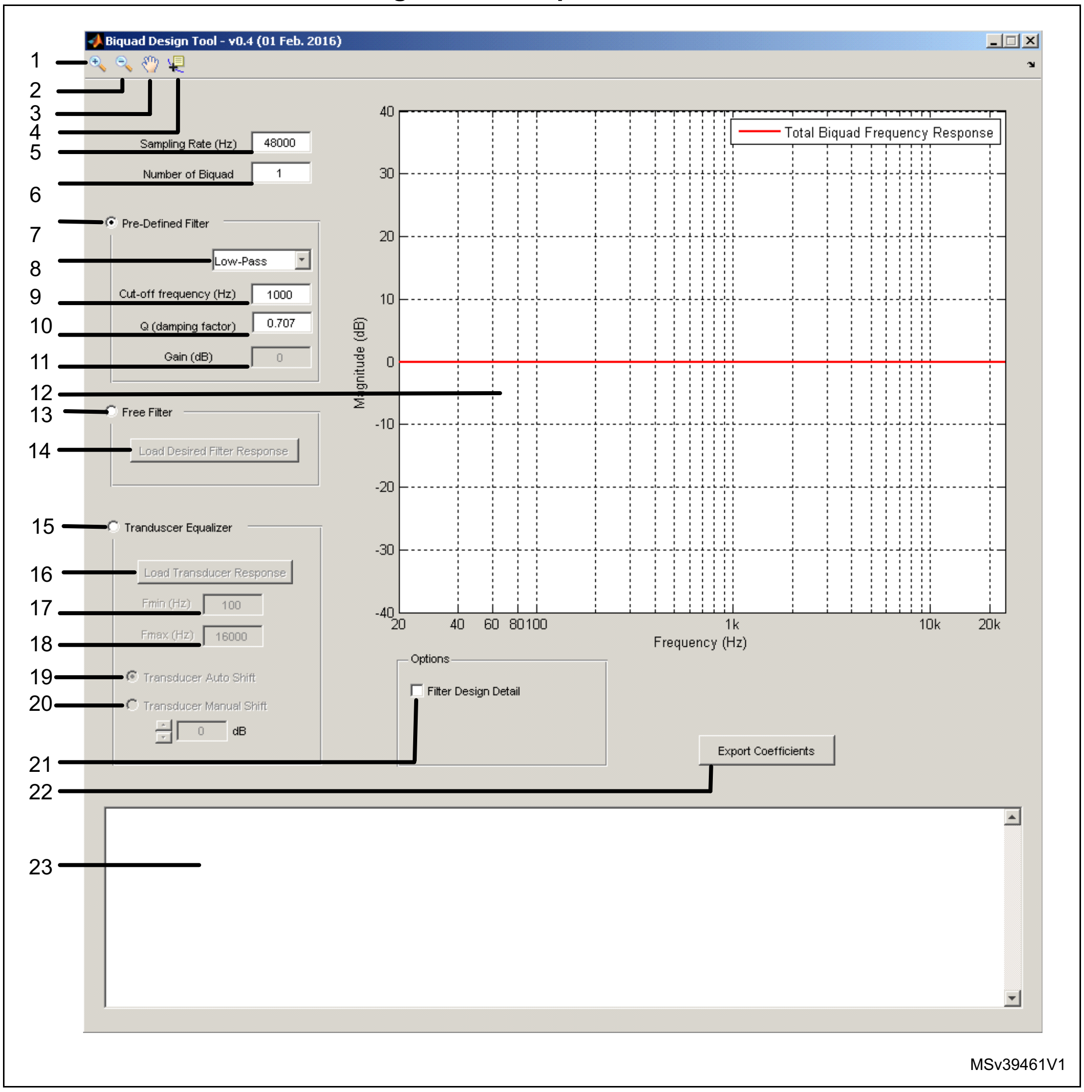


The “Biquad Design Tool” can now be launched.

6.2 Design example

6.2.1 GUI presentation

Figure 14. GUI presentation



1. Zoom-in on plot
2. Zoom-out on plot
3. Enable pan
4. Enable the data cursor on curves
5. Sampling rate (in Hz): it specifies the sampling rate at which the filter will be designed
6. Number of biquad: specifies the number of second order section used for the design. Valid values are within [1 : 10] range.
7. Pre-defined filter: when selected, allows the design of a pre-defined filter type
8. Select the type of pre-defined filter (low-pass, high-pass, peaking, notch, low-shelf, high-shelf)
9. Cut-off frequency (in Hz): specifies the value of the cut-off frequency of the pre-defined filter
10. Q: damping factor of the pre-defined filter. Nominal value is 1/√2 (=0.707)
11. Gain (in dB): it specifies the value of the gain for peaking, low-shelf and high-shelf filter type
12. Axe used for plotting curves
13. Free Filter: when selected, allows the design of an equalizer with an arbitrary frequency response
14. Load desired equalizer: selects and loads the file containing the desired arbitrary frequency response
15. Transducer Equalizer: when selected, allows the design of an equalizer based on a transducer frequency response measurement
16. Load Transducer Response: selects and loads the file containing the transducer frequency response measurement
17. Fmin (in Hz): specifies the minimum valid frequency of transducer frequency response
18. Fmax (in Hz): specifies the maximum valid frequency of transducer frequency response
19. Transducer Auto Shift: when selected, automatically shift the equalizer frequency response magnitude
20. Transducer Manual Shift: when selected, let the user manually shift the equalizer frequency response magnitude
21. Filter Design Detail: When selected, allows plotting of filter design details
22. Export coefficients: select and save the file containing the filter design output. Both floating values, compatible with Matlab and fixed point value, compatible with the Biquad library parameters, are saved
23. Log window: print information

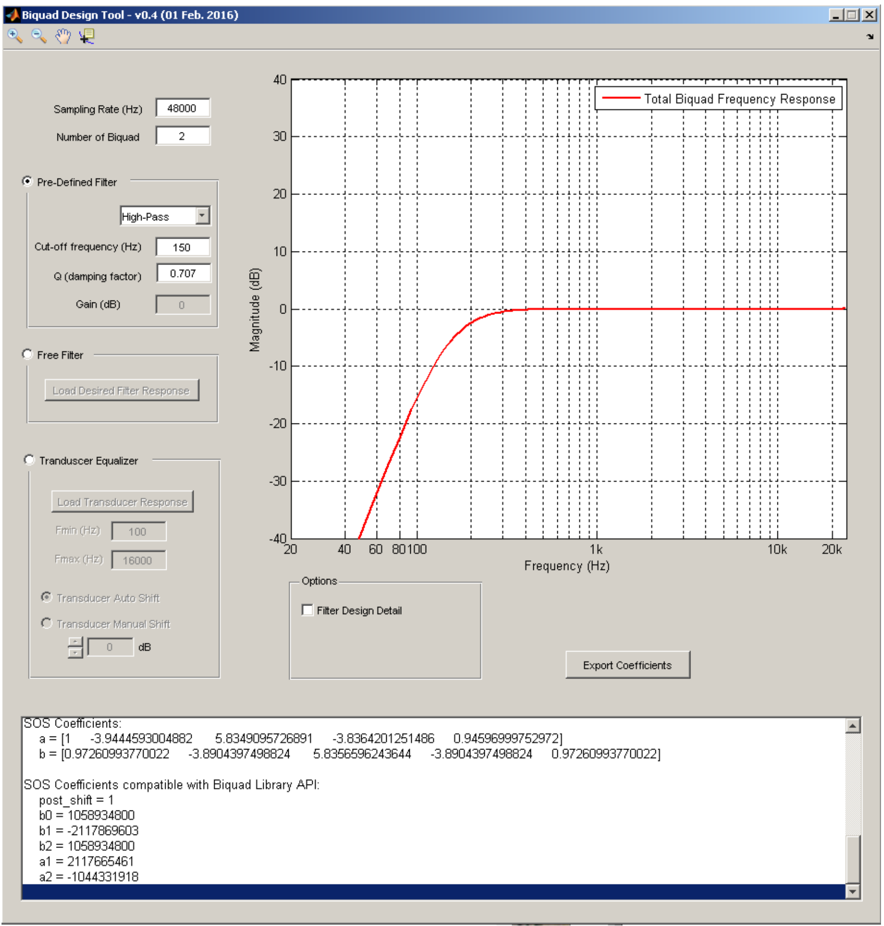
6.2.2 Pre-defined filter

Below is an example of a pre-defined filter design, step by step.

The filter should cut the low frequency, 150Hz , with a roll-off of -24 dB/octave.

The filter is of a “high-pass” type, the sampling rate is equal to 48kHz. The damping factor is equal to 0.707, which is the best trade-off for the transition between the pass-band and stop-band areas. The cut-off frequency is set to 150 Hz. Each Biquad (or second order section) as a slope of -12 dB/octave. Two SOS will then be needed to achieve a slope of -24 dB/octave.

Figure 15. Example of a pre-defined filter



On *Figure 15*, you can see the tool parameters that are set with the previous explanations. The frequency response of the designed filter is computed and plotted out. By using the data cursor tool (see GUI element c), it is possible to check that the filter magnitude at one octave below 150 Hz (i.e. 75 Hz) is at least equal to -24 dB. On the log window, two formats of coefficient are printed:

- [a] and [b] coefficients vector, in floating format. This format can be used as is in Matlab, for example.
- SOS coefficients compatible with Biquad library API. This format is compatible with the parameter structure format, as defined in the *Introduction*. These values can be used in C source code using the Biquad library.

These two coefficient formats can be exported in a text file by clicking on the “export coefficients” button. A filename is required and the coefficients are printed in the following formats:

Figure 16. Coefficients in Matlab format

```
## Generated on 21-3-2013 at 15h21m ##

## Coefficients in Matlab format ##
a = 1      -3.9444593004882      5.8349095726891      -3.8364201251486      0.94596999752972
b = 0.97260993770022      -3.8904397498824      5.8356596243644      -3.8904397498824      0.97260993770022

## Coefficients in Biquad Library format ##
nb_biquad = 2
post_shift = 1
b0 = 1058934800
b1 = -2117869603
b2 = 1058934800
a1 = 2117665461
a2 = -1044331918
post_shift = 1
b0 = 1058934800
b1 = -2117869603
b2 = 1058934800
a1 = 2117665461
a2 = -1044331918
```

6.2.3 Free Filter

Follow the example of a pre-defined filter design, step by step.

The first step is to specify, in a text file, the desired frequency response. *Figure 17* is an example of this input file in the right format:

Figure 17. Input file example

```
0       -48
25      -34
50      -26
100     -18
200     -11
230      -8
300     -15
430     -13
500     -10
590      -7
800     -12
900     -14
1000    -10
2000     -2
4000    -13
6000     -7
8000    -10
12000   -13
15000   -10
24000   -15
```

Each point of the desired frequency response is defined with:

- Its frequency (in Hz). The frequency column must start at 0 Hz and end at half the sampling rate.
- Its magnitude (in dB). As much as possible, avoid using a positive magnitude, as it results in a filter with gains greater than 1 at some frequencies and can lead to clipping.

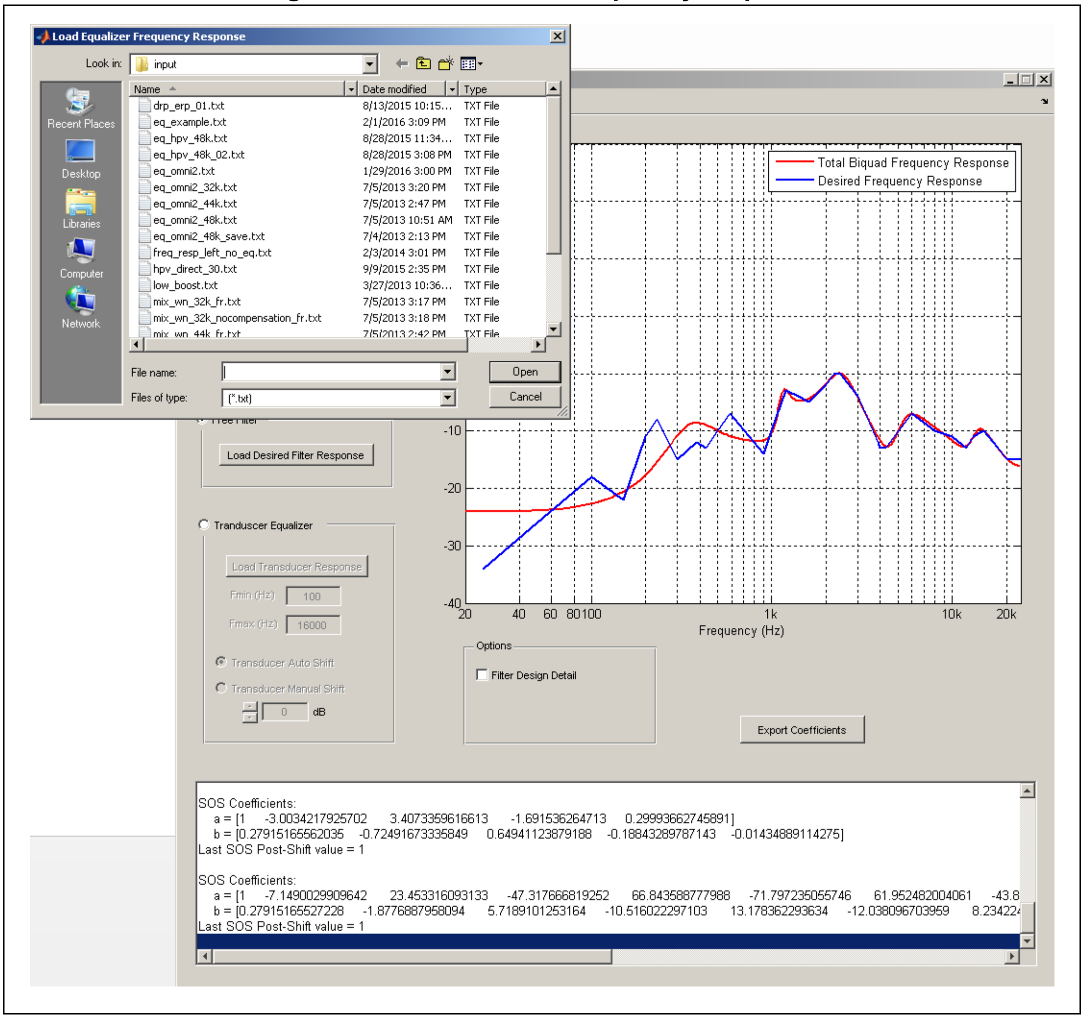
No comments are allowed. There is no restriction to the number of points that can be specified. Nevertheless, having too many points could avoid the tool to converge to the desired frequency response. It is recommended to use logarithmically spaced points.

For example, with n = 3 for a 1/3 octave spacing, frequencies are found using the following formula:

$$Frequency_{next} = Frequency_{previous} * 10^{\frac{3}{10*n}}$$

Once the desired frequency response is specified and saved in a text file, you can load it in the tool by clicking on the “load desired filter response” button:

Figure 18. Load desired frequency response



On the axis, the desired frequency response is plotted in blue. The total Biquad frequency response is plotted in red. The user has now to adjust the number of biquad until he gets an acceptable deviation with the desired response.

On the log window, [a] and [b] coefficient vectors in floating format are printed. This format can be used as it is in Matlab, for example.

The coefficients can be exported in a text file by clicking on the “export coefficients” button. Two coefficient formats are written in the text file, the [a] and [b] coefficients vectors and the SOS coefficients compatible with the Biquad library API. This format is compatible with the parameter structure format, as defined in the *Introduction*. These values can be used in C source code using the Biquad library. *Figure 19* shows the layout of an output example.

Figure 19. Example of coefficient output

```
## Generated on 21-3-2013 at 15h40m ##

## Coefficients in Matlab format ##
a = 1     -3.3003918360354      3.9790651744326     -2.0316176677197      0.35148307928662    -0.015236131120649     0.016806637876297
b = 1.2021393091601     -4.2806704336136      5.7426152784601      -3.467882254256      0.82907822578281    -0.028828512451684     0.003684023398095

## Coefficients in Biquad Library format ##
nb_biquad = 3
post_shift = 1
b0 = 1290787255
b1 = -2086339776
b2 = 860911671
a1 = 1656184695
a2 = -628537230
post_shift = 1
b0 = 1073741824
b1 = -2070284596
b2 = 998963216
a1 = 2021861022
a2 = -950480127
post_shift = 1
b0 = 1073741824
b1 = -17655699
b2 = 5302906
a1 = -134276967
a2 = -34826225
```

6.2.4 Transducer Equalizer

Below is an example of a transducer equalizer design, step by step.

The first step is to have a transducer frequency response measurement, in a text file. Below is an example of this input file in the right format:

Figure 20. Example of transducer equalizer input file

```
0        -30
50        -8
100        0
150       -4
200        7
230       10
300        3
380        6
500        8
800        6
900        4
1000       8
2000      16
3000      14
4000       5
6000      11
8000       8
10000      7
13000      7
18000      5
20000      3
24000      3
```

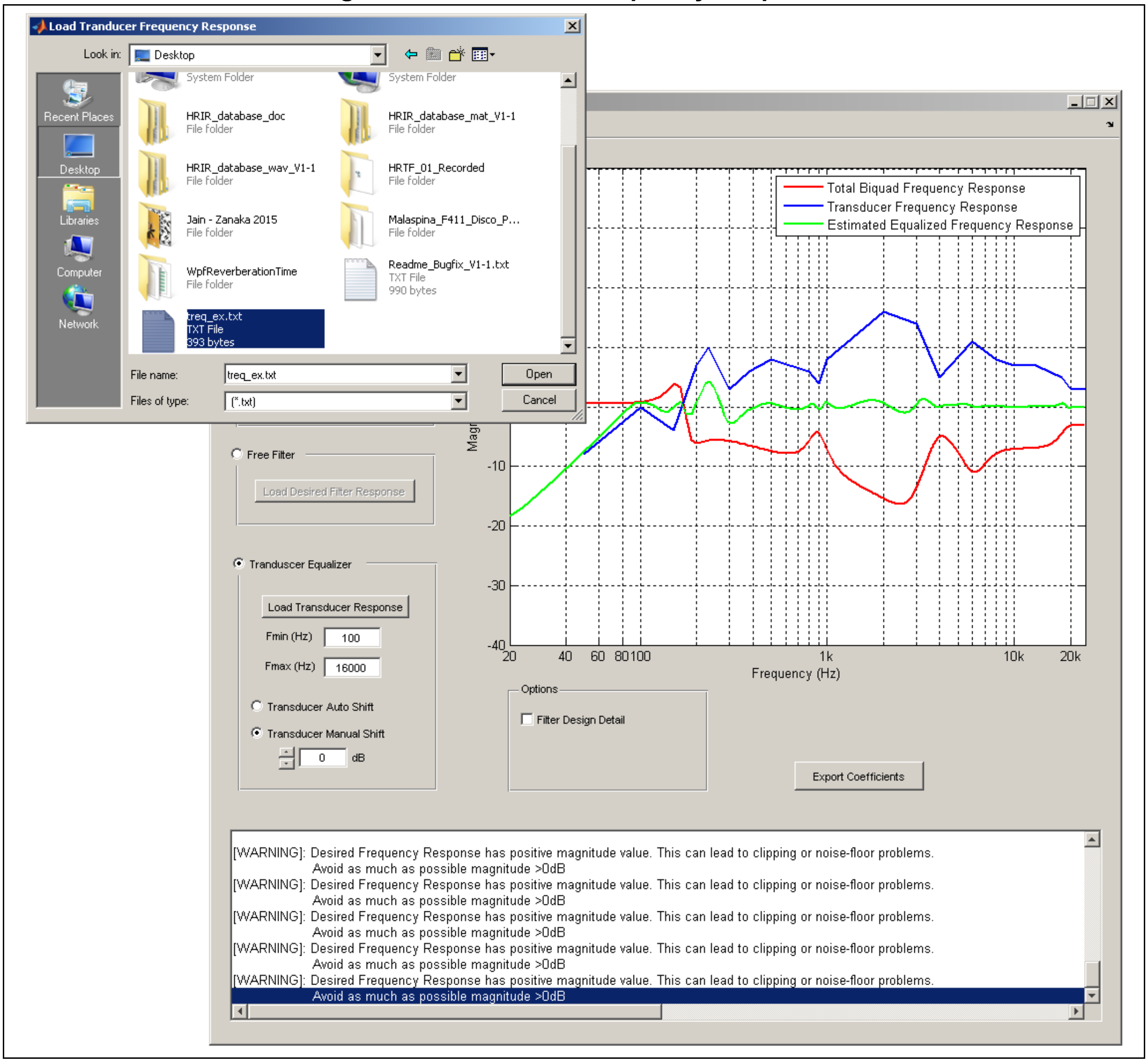
Each point of the transducer frequency response is defined with:

- Its frequency (in Hz). The frequency column must start at 0 Hz and end at half the sampling rate.
- Its magnitude (in dB).

No comments are allowed. There is no restriction to the number of points that can specified. Nevertheless, having too many points could avoid the tool to converge to an optimal equalizer. It is recommended to use logarithmically spaced points.

Once the transducer frequency response has been measured and saved in a text file, one can load it in the tool by clicking on the “load transducer response” button:

Figure 21. Transducer Frequency Response



On the axis, the transducer frequency response is plotted in blue. The total Biquad frequency response is plotted in red. The equalized transducer frequency response estimation is plotted in green.

The user can now adjust different parameters:

- Number of Biquad: from 1 to 10, the more Biquad are used, the more flat the equalized frequency response will be.
- Fmin: specified in Hz, the frequencies below Fmin will not be taken into account in the equalizer computation. For example, if the equalized transducer has a mechanical roll-off starting at 150Hz, it is useless trying to equalize at the frequencies below this value.
- Fmax: specified in Hz, the frequencies above Fmax will not be taken into account in the equalizer computation. For example, if the equalization applies to an acoustic system

with a cut-off frequency of 15kHz, it is not optimal to use Biquad cells for equalization above this frequency.

- Transducer Auto Shift: when used, the transducer frequency response is inverted to get the equalizer frequency response. Then, the maximum of the magnitude, in the range [Fmin : Fmax] is adjusted to 0dB in order to minimize clipping artifacts.
- Transducer Manual Shift: when used, the transducer frequency response is inverted to get the equalizer frequency response. The user can then adjust the magnitude offset manually.

The coefficients can be exported in a text file by clicking on the “export coefficients” button. Two coefficients formats are written in the text file, the transfer function poles ans zeros vectors, and the SOS coefficients compatible with the Biqaud library API. This format is compatible with the parameter structure format, as defined in the introduction.

7 Revision history

Table 13. Document revision history

Date	Revision	Changes
17-Jun-2013	1	Initial release.
26-Nov-2014	2	Classification changed from ST Restricted to public Replaced the reference STSW-STM32APP by STM32-AUDIO100A
07-Mar-2016	3	Updated: – Document title – *Introduction* – *Section 5: How to tune and run the application* – *Section 1.3: Resource summary* – *Section 2.1: APIs* – *Section 4.1.2: Module API calls* – *Section 6.2.3: Free Filter* – *Table 1: Resource summary* – *Figure 7: API call procedure* – *Figure 14: GUI presentation* – *Figure 15: Example of a pre-defined filter* – *Figure 17: Input file example* – *Figure 18: Load desired frequency response* Added: – *Figure 5: Biquad frequency response* – *Section 6.2.4: Transducer Equalizer*
09-Jan-2018	4	Updated: – *Introduction* – *Section 1.2: Module configuration* – *Section 1.3: Resource summary* – *Table 1: Resource summary* – *Section 2: Module Interfaces* – *Section 4.1.1: Module integration example* – *Section 5: How to tune and run the application*