Introduction

The Audio Engine post-processing on STM32F4xx - Biquad library user manual describes the software interface and requirements of the Biquad module.

All necessary interface functions, parameters, integration constraints for the programmer to integrate the Biquad library into his own software are described in this document.

The Biquad library is part of the STM32-AUDIO100A firmware package.
# Contents

1 Module overview ................................................. 6  
  1.1 Algorithm function ........................................ 6  
  1.2 Module configuration ....................................... 6  
  1.3 Resource summary ........................................... 6  

2 Module Interfaces .................................................. 7  
  2.1 APIs .......................................................... 7  
    2.1.1 biquad_df1_cascade_reset function ...................... 7  
    2.1.2 biquad_df1_cascade_setParam function ................... 7  
    2.1.3 biquad_df1_cascade_getParam function ................... 8  
    2.1.4 biquad_df1_cascade_setConfig function .................. 8  
    2.1.5 biquad_df1_cascade_getConfig function .................. 8  
    2.1.6 biquad_df1_cascade_process function .................... 9  
  2.2 External definitions and types .............................. 9  
    2.2.1 Input and output buffers ............................... 9  
    2.2.2 Returned error values .................................. 10  
  2.3 Static parameters structure ................................ 10  
  2.4 Dynamic parameters structure ............................... 11  

3 Algorithm description ............................................. 12  
  3.1 Processing steps ............................................ 12  
  3.2 Data formats ................................................ 13  
  3.3 Performances measurements .................................. 14  
    3.3.1 SNR .................................................... 14  
    3.3.2 Frequency response measurements ....................... 14  

4 Application description ........................................... 16  
  4.1 System requirements and hardware setup ..................... 16  
  4.2 Recommendations for optimal setup .......................... 16  
    4.2.1 Memory allocation ..................................... 16  
    4.2.2 Module API calls ...................................... 17  

5 How to tune and run the application ............................. 19
6 Biquad design tool .................................................. 20
  6.1 Tool installation ................................................. 20
  6.2 Design example .................................................. 23
    6.2.1 GUI presentation ........................................... 23
    6.2.2 Pre-defined filter ......................................... 24
    6.2.3 Free equalizer ............................................. 26

7 Revision history .................................................... 29
List of tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Resource summary</td>
<td>6</td>
</tr>
<tr>
<td>Table 2</td>
<td>biquad_df1_cascade_reset</td>
<td>7</td>
</tr>
<tr>
<td>Table 3</td>
<td>biquad_df1_cascade_setParam</td>
<td>7</td>
</tr>
<tr>
<td>Table 4</td>
<td>biquad_df1_cascade_getParam</td>
<td>8</td>
</tr>
<tr>
<td>Table 5</td>
<td>biquad_df1_cascade_setConfig</td>
<td>8</td>
</tr>
<tr>
<td>Table 6</td>
<td>biquad_df1_cascade_getConfig</td>
<td>8</td>
</tr>
<tr>
<td>Table 7</td>
<td>biquad_df1_cascade_process</td>
<td>9</td>
</tr>
<tr>
<td>Table 8</td>
<td>Input and output buffers</td>
<td>9</td>
</tr>
<tr>
<td>Table 9</td>
<td>Returned error values</td>
<td>10</td>
</tr>
<tr>
<td>Table 10</td>
<td>Static parameters structure</td>
<td>10</td>
</tr>
<tr>
<td>Table 11</td>
<td>biquad_coeff structure</td>
<td>10</td>
</tr>
<tr>
<td>Table 12</td>
<td>Dynamic parameters structure</td>
<td>11</td>
</tr>
<tr>
<td>Table 13</td>
<td>Document revision history</td>
<td>29</td>
</tr>
</tbody>
</table>
List of figures

Figure 1. BIQ module ......................................................... 12
Figure 2. Cascaded BIQ modules .......................................... 13
Figure 3. BIQ module with post-shift parameter .................... 13
Figure 4. SNR .................................................................. 14
Figure 5. Biquad frequency response .................................... 15
Figure 6. API call procedure ................................................ 17
Figure 7. Install the Matlab compiler runtime ....................... 20
Figure 8. Accept the licence agreement .................................. 21
Figure 9. File extraction ...................................................... 21
Figure 10. Installation complete ............................................ 22
Figure 11. Instructions: select the file location ...................... 22
Figure 12. Instructions: launch the application ....................... 23
Figure 13. GUI presentation ................................................ 23
Figure 14. Example of a pre-defined filter ............................... 25
Figure 15. Coefficients in Matlab format ............................... 26
Figure 16. Input file example ................................................ 26
Figure 17. Load desired frequency response ......................... 27
Figure 18. Example of coefficient output ............................... 27
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). The BIQ module can handle mono or stereo/interleaved audio streams.

1.2 Module configuration

The BIQ module supports Mono and Stereo interleaved 16-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.

1.3 Resource summary

Table 1 contains the module requirements for the Flash, stack and RAM memories, and frequency (MHz). The core frequency (MHz) value is an estimation based on the IAR v6.50 simulation profiling.

The frequency (MHz) estimation is for a configuration with 10 biquads, with a stereo audio stream at 48 kHz.

Table 1. Resource summary

<table>
<thead>
<tr>
<th>Flash code (.text)</th>
<th>Flash data (.rodata)</th>
<th>Stack</th>
<th>Static RAM</th>
<th>Dynamic RAM</th>
<th>Frequency (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>562 Bytes</td>
<td>8 Bytes</td>
<td>70 Bytes</td>
<td>600 Bytes</td>
<td>4 Bytes</td>
<td>15.3 MHz</td>
</tr>
</tbody>
</table>
**2 Module Interfaces**

Two files are needed to integrate the BIQ module: *lib_biquad_df1cascade_m4.a* library, and *biquad_df1cascade_glo.h* header file which contains all definitions and structures to be exported to the framework.

*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. They allow the developer to initialize, reset, set and get parameters.

**2.1.1 biquad_df1cascade_reset function**

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

```c
int32_t biquad_df1cascade_reset(void *static_mem_ptr, void *dynamic_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Input</td>
<td>dynamic_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal dynamic memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>-</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

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

**2.1.2 biquad_df1cascade_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.

```c
int32_t biquad_df1cascade_setParam(biquad_df1cascade_static_param_t *input_static_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_static_param_ptr</td>
<td>biquad_df1cascade_static_param_t*</td>
<td>Pointer to static parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>-</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>
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.

```c
int32_t biquad_df1_cascade_getParam(biquad_df1_cascade__static_param_t *input_static_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_static_param_ptr</td>
<td>Biquad_df1_cascade_static_param_t*</td>
<td>Pointer to static parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td></td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

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.

```c
int32_t biquad_df1_cascade_setConfig(biquad_df1_cascade_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_dynamic_param_ptr</td>
<td>Biquad_df1_cascade_dynamic_param_t*</td>
<td>Pointer to dynamic parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td></td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

2.1.5 biquad_df1_cascade_getConfig function

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

```c
int32_t biquad_df1_cascade_getConfig(biquad_df1_cascade_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_dynamic_param_ptr</td>
<td>src236_dynamic_param_t</td>
<td>Pointer to dynamic parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Output</td>
<td></td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>
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.

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

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_buffer</td>
<td>buffer_t *</td>
<td>Pointer to input buffer structure</td>
</tr>
<tr>
<td>Output</td>
<td>output_buffer</td>
<td>buffer_t *</td>
<td>Pointer to output buffer structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Output</td>
<td>int32_t</td>
<td></td>
<td>Error value</td>
</tr>
</tbody>
</table>

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; else, errors will be returned:

```c
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>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_channels</td>
<td>int32_t</td>
<td>Number of channels in data: 1 for mono, 2 for stereo</td>
</tr>
<tr>
<td>nb_bytes_per_Sample</td>
<td>int32_t</td>
<td>16-bit = 2, 24-bit = 3, 32-bit = 4</td>
</tr>
<tr>
<td>data_ptr</td>
<td>void *</td>
<td>Pointer to data buffer (must be allocated by the main framework)</td>
</tr>
<tr>
<td>buffer_size</td>
<td>int32_t</td>
<td>Number of samples per channel in the data buffer</td>
</tr>
<tr>
<td>mode</td>
<td>int32_t</td>
<td>In case of stereo stream, left and right channels can be interleaved. 0 = not interleaved, 1 = interleaved.</td>
</tr>
</tbody>
</table>
2.2.2 Returned error values

Possible returned error values are described below:

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIQ_ERROR_NONE</td>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>BIQ_UNSUPPORTED_INTERLEAVING_MODE</td>
<td>-1</td>
<td>Input data is stereo/not interleaved</td>
</tr>
<tr>
<td>BIQ_UNSUPPORTED_NB_CHANNELS</td>
<td>-2</td>
<td>Input data is neither mono nor stereo</td>
</tr>
<tr>
<td>BIQ_UNSUPPORTED_NB_OF_BYTEPERSAMPLE</td>
<td>-3</td>
<td>Input data is not a 16-bit sample format</td>
</tr>
</tbody>
</table>

2.3 Static parameters structure

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

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

Table 9. Returned error values

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_sos</td>
<td>uint8_t</td>
<td>Number of second order sections for the current configuration. Valid range is [1:10]</td>
</tr>
<tr>
<td>biquad_coff[]</td>
<td>int32_t</td>
<td>Array of SOS coefficients and post-shift values. It contains 6 parameters for each SOS. See Table 11: biquad_coff structure. For each SOS, coefficient values are computed with the following formula: [ \text{coeff}<em>{\text{int32}} = \text{round}(\text{coeff}</em>{\text{dec}} \times 2^{31 - \text{post-shift}}) ]</td>
</tr>
</tbody>
</table>

Table 10. Static parameters structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_shift</td>
<td></td>
<td>SOS coefficient and post-shift values</td>
</tr>
<tr>
<td>b0</td>
<td></td>
<td>1st SOS</td>
</tr>
<tr>
<td>b1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>b2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
2.4 Dynamic parameters structure

It is possible to change the Biquad configuration by setting the new coefficient 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.

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

Table 11. biquad_coeff structure

<table>
<thead>
<tr>
<th>SOS coefficient and post-shift values</th>
<th>SOS</th>
</tr>
</thead>
<tbody>
<tr>
<td>post_shift</td>
<td></td>
</tr>
<tr>
<td>b0</td>
<td>2nd SOS</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td>Last SOS</td>
</tr>
<tr>
<td>post_shift</td>
<td></td>
</tr>
<tr>
<td>b0</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>a2</td>
<td></td>
</tr>
</tbody>
</table>

Table 12. Dynamic parameters structure

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enable</td>
<td>uint8_t</td>
<td>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</td>
</tr>
<tr>
<td>nb_sos</td>
<td>uint8_t</td>
<td>Number of second order sections for the current configuration. Valid range is [1:10]</td>
</tr>
<tr>
<td>biquad_coeff[]</td>
<td>int32_t</td>
<td>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: $coeff_{int32} = \text{round}(coeff_{dec} \times 2^{31 - \text{post}_{shift}})$</td>
</tr>
</tbody>
</table>
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:

\[ a_0 \times y(n) = b_0 \times x(n) + b_1 \times x(n - 1) + b_2 \times x(n - 2) - a_1 \times y(n - 1) - a_2 \times y(n - 2) \]

All coefficients are normalized to \(a_0\), so that the final equation has 5 coefficients:

\[ y(n) = \frac{b_0}{a_0} \times x(n) + \frac{b_1}{a_0} \times x(n - 1) + \frac{b_2}{a_0} \times x(n - 2) - \frac{a_1}{a_0} \times y(n - 1) - \frac{a_2}{a_0} \times y(n - 2) \]

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. \(a_1\) and \(a_2\) 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:
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:

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, else 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-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 fed with full-scale sinusoid at different frequencies. The SNR measurement is then performed at different frequencies and compared to the input signal. Below is the plot of input/output SNR comparison:

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

The SNR formula used is:

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

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 2\textsuperscript{nd} 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 System requirements and hardware setup

The library is built to run on a Cortex® M4 core without FPU usage. It was integrated and validated on some STM32F4 family devices.

4.2 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. Once this is done, the inverted speaker frequency response can be used as the input (the desired frequency response) in the tool. Be careful not to put too much positive gain in the desired frequency response.

4.2.1 Memory allocation

The static and dynamic parameter structures must be allocated. Their types are defined in `biquad_df1_cascade_glo.h` header.

Example of an allocation:

```c
/* parameters structure memory allocation */
biquad_df1_cascade_static_param_t *static_param_ptr = alloc(sizeof(biquad_df1_cascade_static_param_t));
biquad_df1_cascade_dynamic_param_t *dynamic_param_ptr = malloc(sizeof(biquad_df1_cascade_dynamic_param_t));
```

The static and dynamic memory pointer must be allocated too. The size of each one is defined in `biquad_df1_cascade_glo.h` header.

Example of an allocation:

```c
/* memory structure memory allocation */
void *static_mem_ptr = malloc(biquad_df1_cascade_static_mem_size);
void *dynamic_mem_ptr = malloc(biquad_df1_cascade_dynamic_mem_size);
```

It is then necessary to allocate the memory for input and output audio buffers.
4.2.2 Module API calls

Figure 6. API call procedure

1. Memory allocation
2. `biquad_df1_cascade_reset()`
3. `static_param initialization`
4. `biquad_df1_cascade_setParam()`
5. `audio stream read`
   `input_buffer preparation`
6. `biquad_df1_cascade_process()`
7. `audio stream write`

New Config needed?

8. `biquad_df1_cascade_setConfig()`
9. `Memory freeing`
1. As explained above, Biquad static and dynamic 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 has been 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. Once the processing loop is over, the 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 Biquad library.

Three different settings have been pre-computed and can be selected, using one of the following check boxes:

- **Woofer**: Sets a 4th order low-pass filter with a cut-off frequency equals to 150 Hz.
- **Medium/High**: Sets a 4th order high-pass filter with a cut-off frequency equals to 150 Hz.
- **Loudness**: Sets an approximation of an equal-loudness frequency response curve (A-weighing curve).
  
  **Caution**: this configuration provides positive gain at some frequencies and can lead to some clipping artifacts with a high amplitude test signal.

One additional check box is used to enable/disable the effect:

- **Enable**: enable/disable the effect.
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 has been 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 7. Install the Matlab compiler runtime
Figure 8. Accept the licence agreement

License Agreement

The MathWorks, Inc.

License Agreement

License Agreement

Do you accept the terms of the license agreement? Yes No

Figure 9. File extraction

File extraction

File extraction

File extraction

About 10 minutes remaining

File extraction

File extraction

File extraction
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_v1.0.exe` file and follow the instructions:
Figure 12. Instructions: launch the application

The “Biquad Design Tool” can now be launched.

6.2 Design example

6.2.1 GUI presentation

Figure 13. GUI presentation
1. Zoom-in on the plot
2. Zoom-out on the plot
3. Enables the data cursor on curves
4. Axe used for plotting curves
5. Sampling rate (in Hz): specifies the sampling rate at which the filter will be designed.
6. Number of biquad: specifies the number of second order sections 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. Selects 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 $\frac{1}{\sqrt{2}} (=0.707)$
11. Gain (in dB): specifies the value of the gain for peaking, low-shelf and high-shelf filter type
12. Free equalizer: when selected, allows the design of an equalizer with an arbitrary frequency response
13. Load desired equalizer: selects and loads the file containing the desired arbitrary frequency response
14. Export coefficients: selects and saves 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.
15. 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.
On Figure 14, 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 15. Coefficients in Matlab format

```matlab
%% Generated on 31-3-2015 at 15h28m XX

%% Coefficients in Matlab format %
% a = 1 -3.04342020020002 5.86565606560656
% b = 0.28049707900000 0.28049707900000
% db = 1 0

%% Coefficients in Biquad Library format %
% b0 = 106034000
% b1 = -211766461
% b2 = 104431910
% a0 = 1
% a1 = 0
```

### 6.2.3 Free equalizer

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 16* is an example of this input file in the right format:

#### Figure 16. Input file example

<table>
<thead>
<tr>
<th>Frequency (Hz)</th>
<th>Magnitude (dB)</th>
<th>Weight</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.0</td>
<td>0</td>
<td>0.01</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td>0.5</td>
<td>1</td>
</tr>
<tr>
<td>125</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>2000</td>
<td>-2</td>
<td>1</td>
</tr>
<tr>
<td>3000</td>
<td>-2.5</td>
<td>1</td>
</tr>
<tr>
<td>5000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7000</td>
<td>-3</td>
<td>1</td>
</tr>
<tr>
<td>10000</td>
<td>-2.5</td>
<td>1</td>
</tr>
<tr>
<td>14000</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>20000</td>
<td>-1.5</td>
<td>1</td>
</tr>
<tr>
<td>60000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>120000</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>113300</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>111100</td>
<td>3</td>
<td>1</td>
</tr>
<tr>
<td>214000</td>
<td>3</td>
<td>1</td>
</tr>
</tbody>
</table>

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.
- **Its weight (default = 1).** Weight values are relative to each other. 1 is the reference. A weight value of 0.1 indicates to the tool that the filter response has not constraints to reach this point. A weight value of 10 indicates that this point must be reached.

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 advised to use logarithmically spaced points.
For example, with \( n = 3 \) for a 1/3 octave spacing, frequencies are found using the following formula:

\[
\text{Frequency}_{\text{next}} = \text{Frequency}_{\text{previous}} \times 10^{\frac{3}{n}}
\]

Once the desired frequency response has been specified and saved in a text file, you can load it in the tool by clicking on the "load desired frequency response" button:

![Figure 17. Load desired frequency response](image)

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 18 shows the layout of an output example.

![Figure 18. Example of coefficient output](image)
## Biquad design tool UM1625

**Coefficients in Biquad format**

```
Q = 1  -0.3039103458489  -3.379681715834  -2.031174715787  0.35118717502545  -0.03521631120596  0.004204079707019
R = 1.0021391308150  -4.20067080336325  3.7421581784801  -2.467882546455  0.83078125706280  -0.82628292481684  0.00346052239895

**Coefficients in Biquad Library format**

```
Q2 = 1
G = 1
A0 = 1.0021391308150
A1 = 0
A2 = 0
B0 = -0.3039103458489
B1 = 0
B2 = 0
C0 = 3.7421581784801
C1 = 0
C2 = 0
D0 = 2.467882546455
D1 = 0
D2 = 0
```
## Revision history

Table 13. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>17-Jun-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>26-Nov-2014</td>
<td>2</td>
<td>Classification changed from ST Restricted to public</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Replaced the reference STSW-STM32APP by STM32-AUDIO100A</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST’s terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers’ products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2014 STMicroelectronics – All rights reserved