# UM1798
# User manual

## Audio engine post-processing on STM32F4xx Graphical Equalizer library

## Introduction

This document reviews the software interface and requirements of the Graphic Equalizer (GrEQ) module.

All necessary interface functions, parameters, integration constraints for the programmer to integrate the GrEQ library (STM32-AUDIO100A) into his own software are described in this document.

The GrEQ library is designed to run on an ARM® Cortex®-M4 core without FPU usage, so it can be integrated and run on all the STM32F40xxx, STM32F41xxx, STM32F42xxx or STM32F43xxx products.

# Contents

# List of tables

# List of figures

# 1 Module overview

## 1.1 Algorithm functionality

The GraphicEQualizer (GrEQ) module is in charge of fine tuning the sound spectrum according to user personal preferences. This is done by modifying gain factors at fixed frequencies represented by sliders.

The number of bands is determined at the initialization phase and can be 5, 8 or 10. The gain factors are adjustable from -12dB to +12dB in standard mode. The library can be generated with different maximum gains (+18dB, +24dB).

The current implementation uses 32 bits resolution for all computations, and can be used with both 16 and 32 bits input/output format.

## 1.2 Module configuration

GrEQ module supports Mono and Stereo interleaved 16 or 32 bits I/O data, with a maximum input frame size of 480 stereo samples. This limitation corresponds to 10ms scheduling at 48 kHz sampling frequency.

## 1.3 Resources summary

*Table 1* contains CPU, Flash, Stack and RAM requirements. The core MHz value is an estimation based on IAR systems® v6.50 simulation profiling.

**Table 1. Summary of resources**

| Flash code (.text) | Flash data (.rodata) | Stack | Static RAM | Dynamic RAM | CPU frequency |
|---|---|---|---|---|---|
| - | - | - | - | - | Stereo/interleaved, 48 kHz |
| 4344 Bytes | 8 Bytes | 92 Bytes | 548 Bytes | 3840 Bytes | 14.5 MHz |

# 2 Module interfaces

Two files are needed to integrate the GrEQ module. *lib_greq_m4.a* library supports 16 bits I/O data, while *lib_greq_m4_32b.a* library supports 32 bits I/O data. *greq_glo.h* header file contains all definitions and structures to be exported to the application SW.

Note also that *audio_fw_glo.h* file is a generic header file common to all audio modules and 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 or get parameters and process audio buffers.

### 2.1.1 Function greq_reset

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

```
int32_t greq_reset(void *static_mem_ptr, void *dynamic_mem_ptr);
```

**Table 2. Parameters for function greq_reset**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Input** | *dynamic_mem_ptr* | *void ** | Pointer to internal dynamic 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 yet started.

### 2.1.2 Function greq_setParam

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

```
int32_t greq_setParam(greq_static_param_t *input_static_param_ptr, void
*static_mem_ptr);
```

**Table 3. Parameters for function greq_setParam**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_static_param_ptr* | *greq_static_param_t** | Pointer to static parameters structure |
| **Input** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Returned value** | - | *int32_t* | Error value |

### 2.1.3 Function greq_getParam

This procedure gets the module's static parameters from the module's internal memory to main framework.

It can be called after reset routine and before real time processing started. It handles static parameters (i.e. the parameter values that cannot be changed during module processing).

```
int32_t greq_setParam(greq_static_param_t *input_static_param_ptr, void
*static_mem_ptr);
```

**Table 4. Parameters for function greq_getParam**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_static_param_ptr* | *greq_static_param_t** | Pointer to static parameters structure |
| **Input** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Returned value** | - | *int32_t* | Error value |

### 2.1.4 Function greq_setConfig

This procedure sets module dynamic parameters from main framework to module internal memory.

It can be called at any time during processing.

```
int32_t greq_setConfig( greq_dynamic_param_t *input_dynamic_param_ptr, void
*static_mem_ptr);
```

**Table 5. Parameters for function greq_setConfig**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_dynamic_param_ptr* | *greq_dynamic_param_t** | Pointer to dynamic parameters structure |
| **Input** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Returned value** | - | *int32_t* | Error value |

### 2.1.5 Function greq_getConfig

This procedure gets the module dynamic parameters from internal static memory to the main framework.

It can be called at any time during processing.

```
int32_t greq_getConfig(greq_dynamic_param_t *input_dynamic_param_ptr, void
*static_mem_ptr);
```

**Table 6. Parameters for function greq_getConfig**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_dynamic_param_ptr* | *greq_dynamic_param_t** | Pointer to dynamic parameters structure |
| **Input** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Returned value** | - | *int32_t* | Error value |

### 2.1.6 Function greq_process

This procedure is the module's main processing routine.

It should be called at any time, to process each frame.

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

**Table 7. Parameters for function greq_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** | *static_mem_ptr* | *void ** | Pointer to internal static memory |
| **Output** | - | *int32_t* | Error value |

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

## 2.2 External definitions and types

### 2.2.1 Input and output buffers

The GrEQ library uses 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 respected each time before calling the processing routine; else, errors 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:<br>1 for mono, 2 for stereo, 4 for 3.1 multichannel,... |
| *nb_bytes_per_Sample* | *int32_t* | Dynamic of data in number of bytes:<br>16 bits = 2, 24 bits = 3, 32 bits = 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:<br>0 = not interleaved, 1 = interleaved. |

### 2.2.2 Returned error values

*Table 9* lists possible returned error values:

**Table 9. Error values**

| Definition | Value | Description |
|---|---|---|
| GREQ_ERROR_NONE | 0 | No error |
| GREQ_UNSUPPORTED_NB OF BYTEPERSAMPLES | -1 | Input data is not 16 bit sample format |
| GREQ_UNSUPPORTED_NB_CHANNELS | -2 | Input data is neither mono nor stereo |
| GREQ_UNSUPPORTED_NB_OF_BANDS | -3 | Number of bands not supported |
| GREQ_UNSUPPORTED_GAIN_PRESET | -4 | Gain preset index not supported |
| GREQ_UNSUPPORTED_INTERLEAVING_MODE | -5 | Input data is stereo / not interleaved |
| GREQ_UNSUPPORTED_FRAME_SIZE | -6 | Frame size not supported |
| GREQ_UNSUPPORTED_GAIN | -7 | Gain not supported |
| GREQ_BAD_HW | -8 | Unsupported HW for the library |

## 2.3 Static parameters structure

The GrEQ initial parameters are set using the corresponding static parameter structure before calling the *greq_setParam()* function.

```
struct greq_static_param {
    int16_t  nb_bands;
}
```

**Table 10. nb_bands parameter**

| Name | Type | Description | nb_bands | Center frequencies |
|---|---|---|---|---|
| *nb_bands* | *int16_t* | Defines the number of bands (5, 8 or 10) | 5 | 125, 400, 1278, 4088, 13074 |
| | | | 8 | 100, 203, 412, 837, 1698, 3447, 6998, 14206 |
| | | | 10 | 62, 115, 214, 399, 742, 1380, 2567, 4775, 8882, 16520 |

## 2.4 Dynamic parameters structure

It is possible to change the GrEQ configuration by setting new values in the dynamic parameter structure before calling the *greq_setConfig()* function.

```
struct greq_dynamic_param {
    int16_t   enable;
    int16_t   user_gain_per_band_dB[];
    int16_t   gain_preset_idx;
}
```

**Table 11. Dynamic parameters**

| Name | Type | Description |
|---|---|---|
| *enable* | int16_t | 0: disable the effect, output buffer is equal to input buffer<br>1: enable the effect, output buffer is equal to input buffer processed by GrEQ |
| *user_gain_per_band_dB[10]* | int16_t | Sets the gain for each band.<br>The values are specified in dB, and go from -12 to +12dB in 1dB steps.<br>Depending on the “nb_bands” static parameter value (see *Table 10*), only values 5, 8 or 10 can be selected. |
| *gain_preset_idx* | int16_t | GrEQ library is configured with 6 gain presets. When the user selects one preset, the gains in the “user_gain_per_band_dB[]” table are discarded.<br>0: no preset, use the gains in the “user_gain_per_band_dB[]” table<br>1: Pop<br>2: Jazz<br>3: Rock<br>4: Vocal<br>5: Classical<br>6: HipHop |

# 3 Algorithm description

## 3.1 Processing steps

The GrEQ algorithm is based on the parallelization of band-pass filters (BPFs).

The BPFs are characterized by their center frequencies and Q factor. Depending on the number of bands (5, 8 or 10), the center frequencies and Q factor are set so that BPFs cover the audible spectrum, with each center frequency equally spaced on a logarithmic scale.

The GrEQ embeds a special processing which reduce drastically the neighboring effect of a band on the others. Each band gain can thus be set quasi-independently from the others. Each BPF is implemented using an optimized bi-quadratic structure, using 32 bits coefficients.

## 3.2 Data formats

Input of GrEQ module is expected to be an audio stream, stereo/interleaved, in 16 or 32 bits format. All operations are done with 32 bits resolution. The output format is an audio stream, stereo/interleaved signal in 16 or 32 bits format.

### 3.2.1 Preset frequency responses

The frequency responses for each music style are shown in dedicated figures, as summarized in *Table 12*.

**Table 12. Frequency responses for different presets**

| Preset | Music style | Frequency response |
|---|---|---|
| 1 | Pop | *Figure 1* |
| 2 | Jazz | *Figure 2* |
| 3 | Rock | *Figure 3* |
| 4 | Vocal | *Figure 4* |
| 5 | Classical | *Figure 5* |
| 6 | HipHop | *Figure 6* |

.

**Figure 1. Frequency response for Pop music**

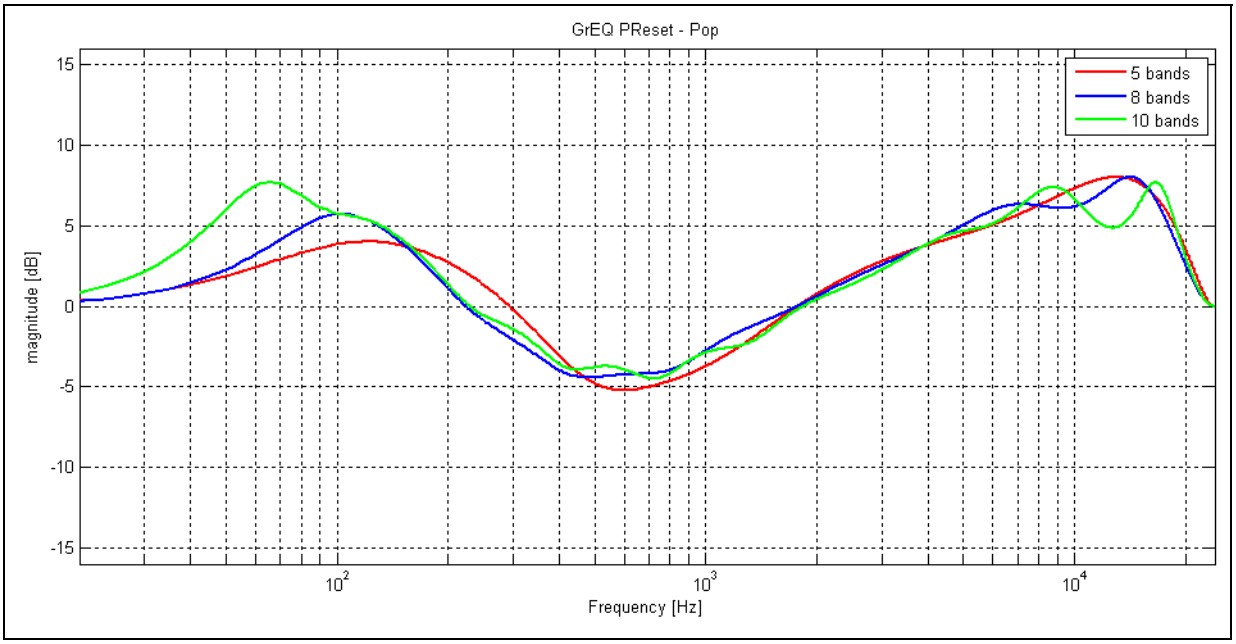



**Figure 2. Frequency response for Jazz music**

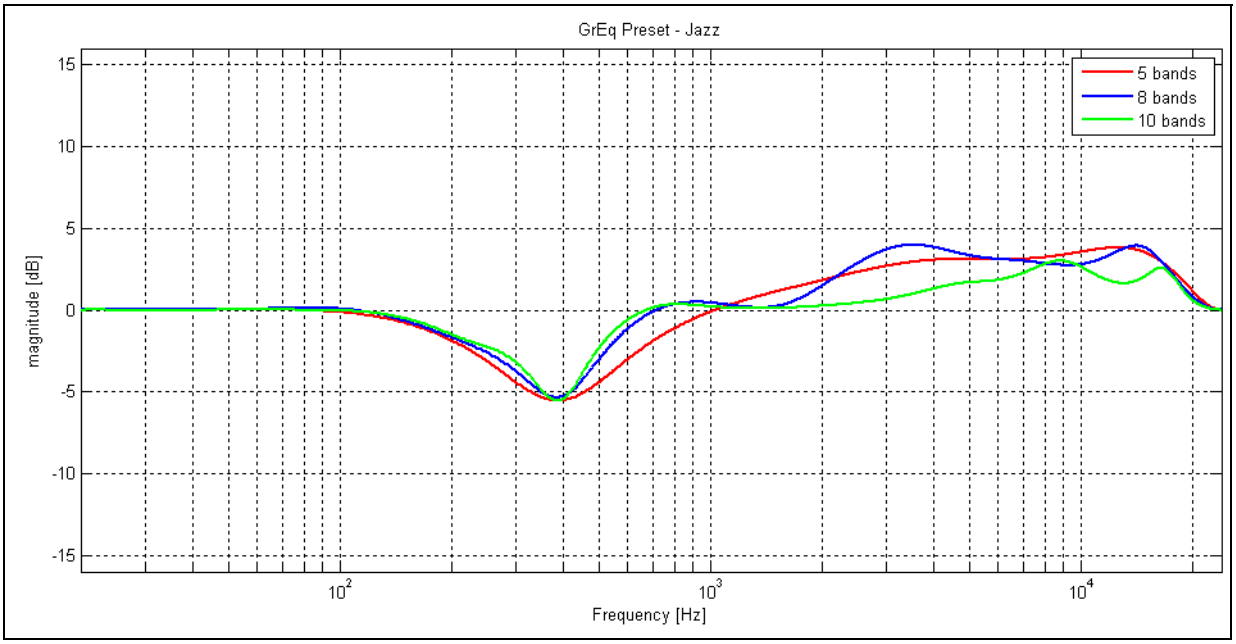


**Figure 3. Frequency response for Rock music**

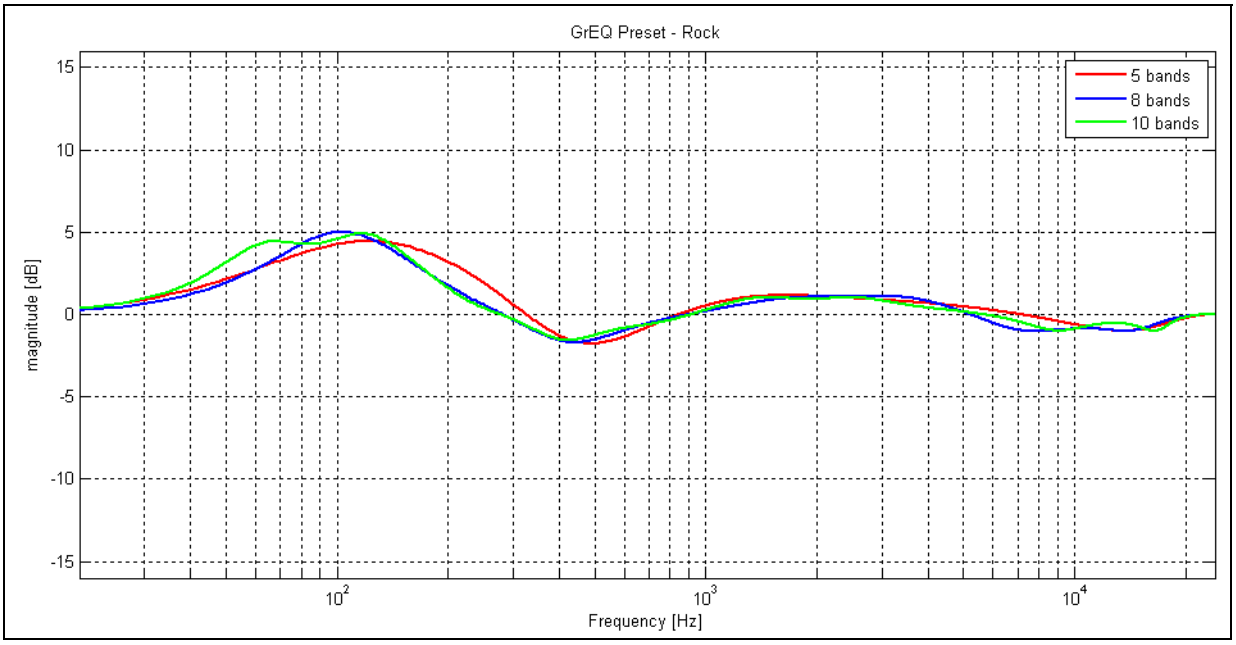



**Figure 4. Frequency response for Vocal music**

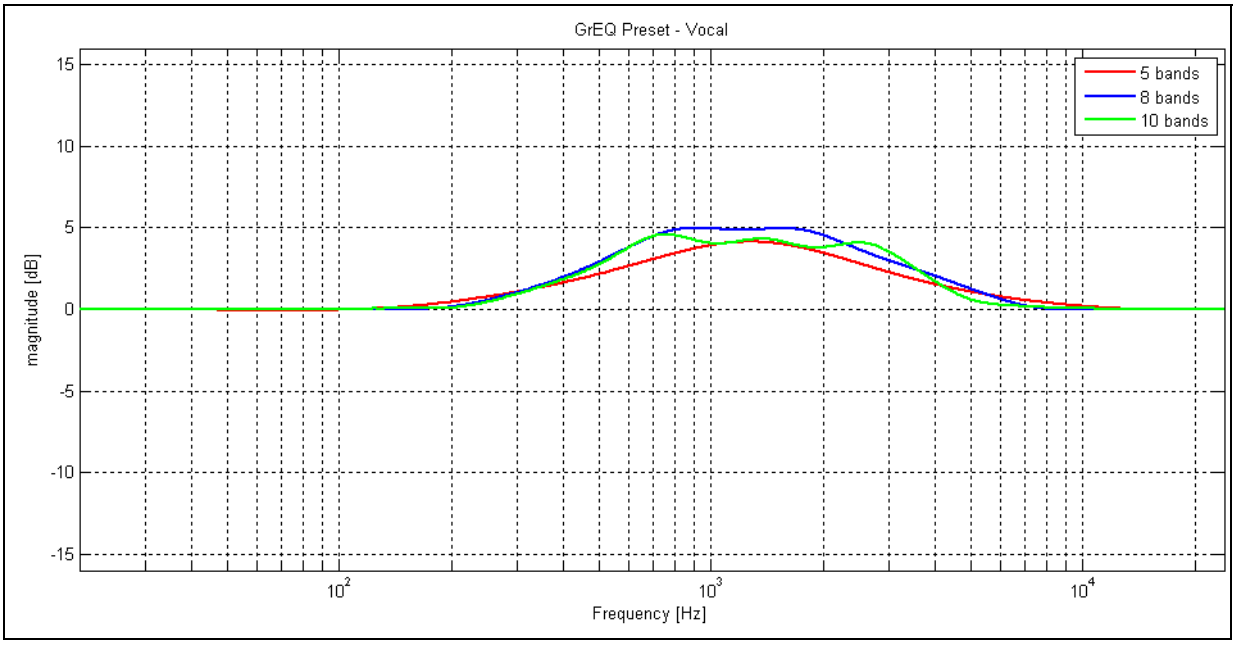


Figure 5. Frequency response for Classical music

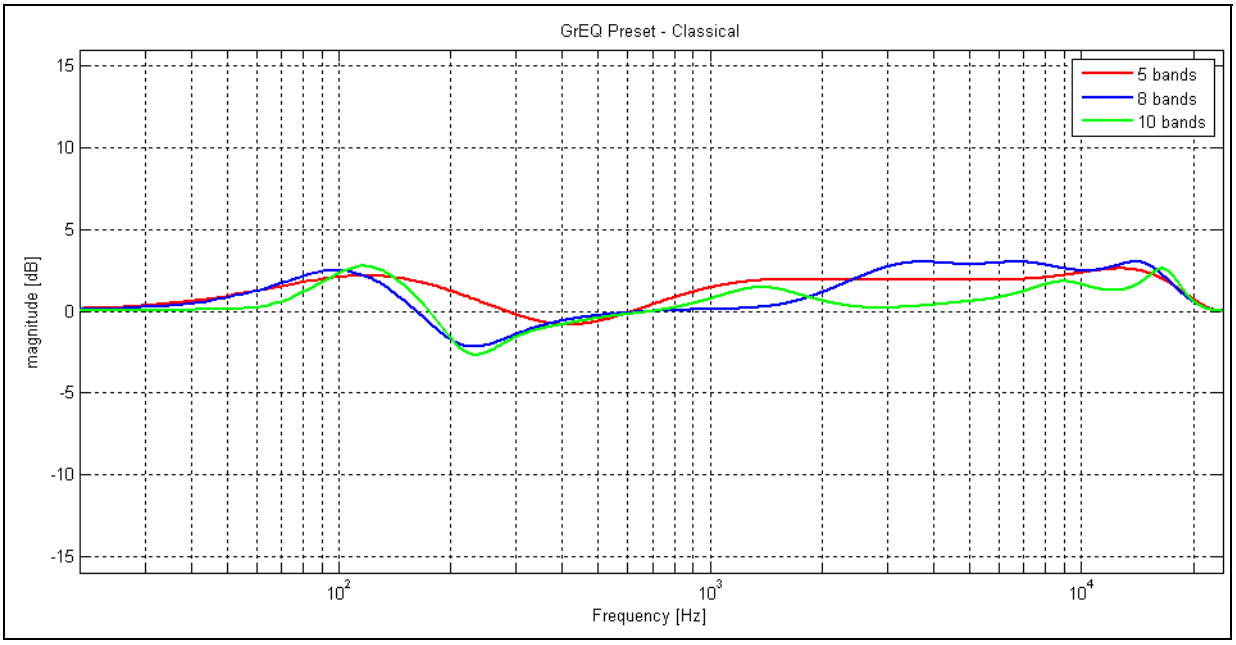
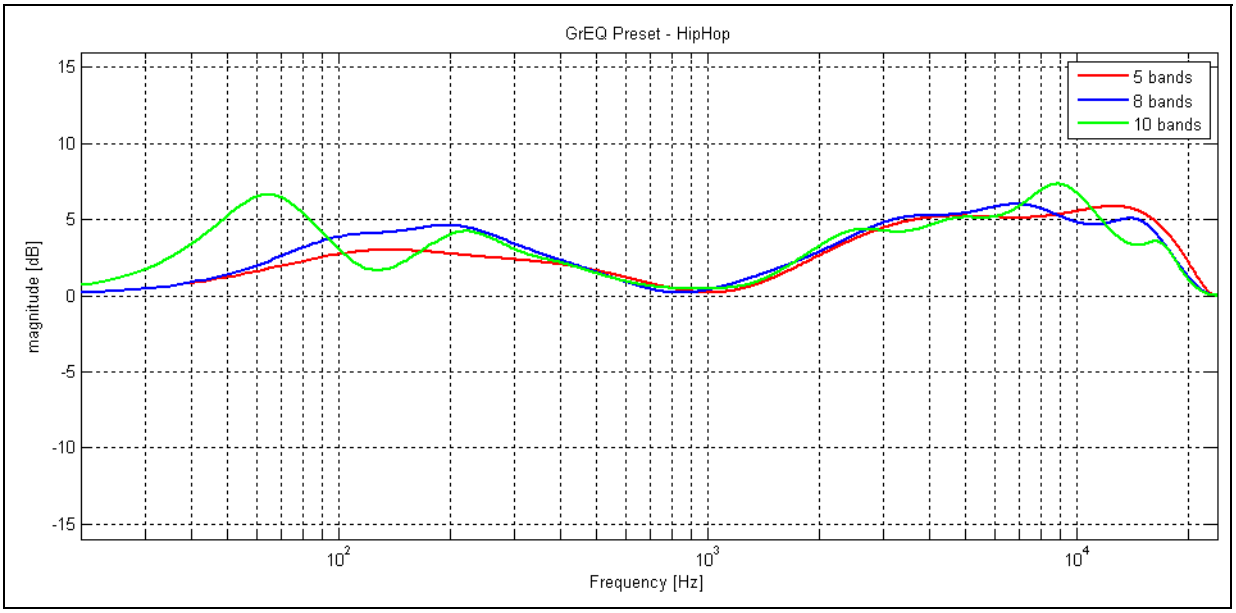



Figure 6. Frequency response for Hip Hop music



# 4 Application Description

## 4.1 Recommendations for Optimal Setup

The GrEQ module can be executed at any place in an audio processing chain (because it applies a linear algorithm, there is no restriction on the order of execution).

However, care should be taken in the gain distribution all over the processing chain, as the GrEQ applies pre-attenuation. At the input of the GrEQ, some margin is taken in order to avoid any saturation when the user is setting positive gains. The default available library is generated with a maximum gain of 12dB, as a consequence 2 bits of guard are taken.

### 4.1.1 Memory allocation

The static and dynamic parameters structures must be allocated. Their types are defined in *greq_glo.h* header. Example of allocation:

```
/* parameters structure memory allocation */
greq_static_param_t *static_param_ptr = malloc(sizeof(greq_static_param_t));
greq_dynamic_param_t *dynamic_param_ptr = malloc(sizeof(greq_dynamic_param_t));
```

The static and dynamic memory pointer must be allocated too. The size of each is defined in *greq_glo.h* header. Example of allocation:

```
/* memory structure memory allocation */
void *static_mem_ptr = malloc(greq_static_mem_size);
void *dynamic_mem_ptr = malloc(greq_dynamic_mem_size);
```

It is then needed to allocate the memory for input and output audio buffers.

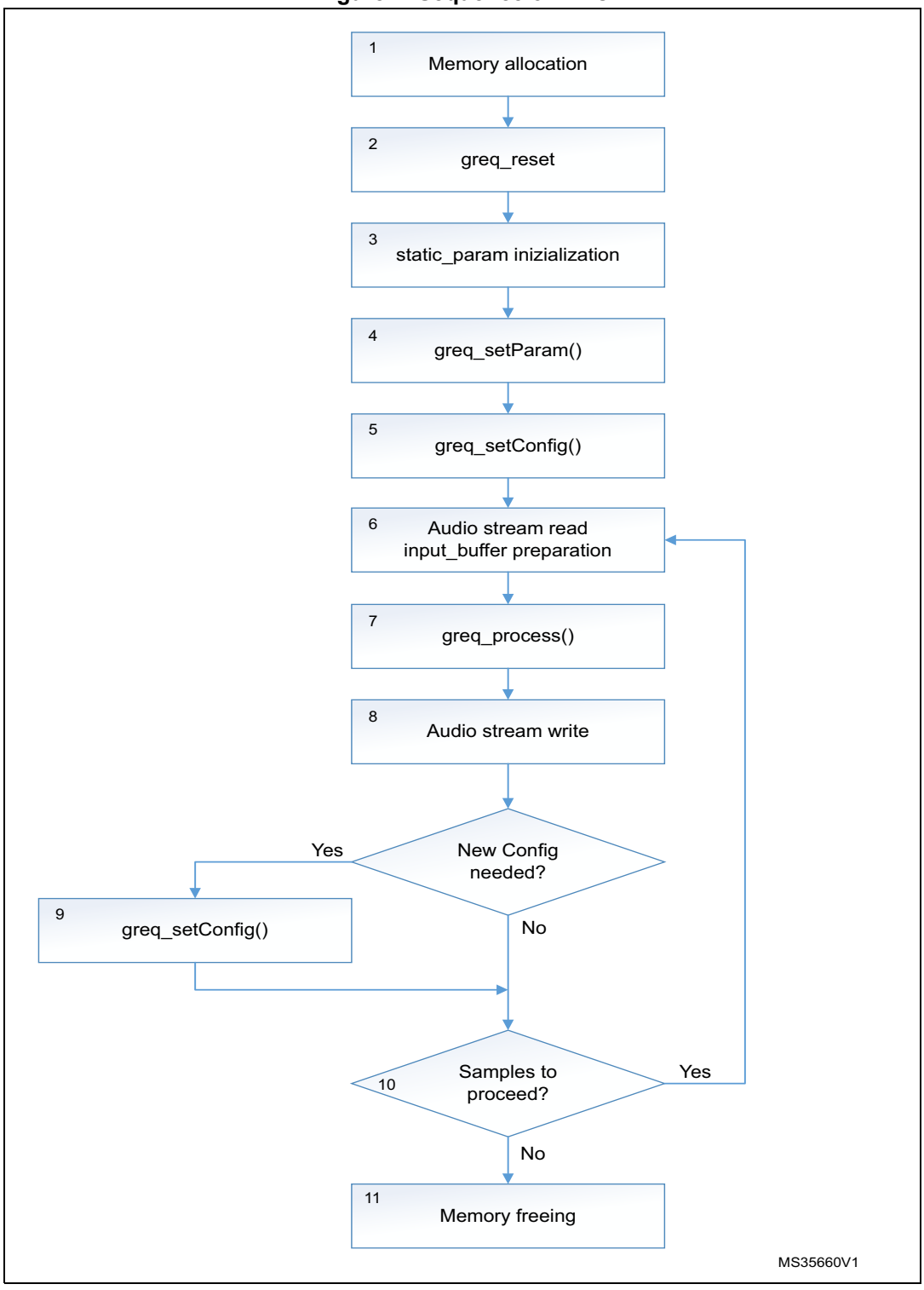
### 4.1.2 Module APIs calls

The sequence is shown in *Figure 7*, and each step is described in detail in the following list:

1. As explained above, GrEQ static and dynamic structures have to be allocated, as well as input and output buffer accordingly to the structures defined in *Section 2.2.1*.
2. Once memory has been allocated, the call to *greq_reset()* function will initialize internal variables.
3. The GrEQ configuration for the desired filter response can now be set by initializing the static_param structure.
4. Calling the *greq_setParam()* function will then configure the GrEQ internal memory according to the desired number of bands.
5. Then the gains per band or the preset can be changed by setting the dynamic parameters structure and calling *greq_setConfig()* function.
6. The audio stream is read from the proper interface and input_buffer structure has to be filled according to the stream characteristics (number of channels, sample rate, interleaving and data pointer). Output buffer structure has to be set as well.
7. Calling the *greq_process()* function will execute the GrEQ algorithm.
8. The output audio stream can now be written in the proper interface.
9. If needed, the user can set new dynamic parameters and call the *greq_setConfig()* function to update module configuration.

10. If the application is still running and has new input samples to proceed, then it goes back to step 6, else the processing loop is over.
11. Once the processing loop is over, allocated memory has to be freed.

**Figure 7. Sequence of APIs**



# 5 How to tune and run the application

The STM32 audio framework provides an example application for the GrEQ library. It uses the library with 10 bands, and typically implements 10 sliders which allow gain adjustment for each band. It also proposes a list-box for selecting one of the six presets.

Tuning of the GrEQ depends entirely on the loudspeaker, user preference and music style. There is no real recommendation for tuning.

# 6 Revision history

Table 13. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 30-Sep-2014 | 1 | Initial release. |