UM1798
User manual

Graphical equalizer library software expansion for STM32Cube

Introduction

This document reviews the software interface and requirements for the Graphical equalizer library. It describes how to integrate the Graphical equalizer module into a main program like the Audio STM32Cube expansion software, and provides a rough understanding of the underlying algorithm.

The Graphical equalizer library is part of the X-CUBE-AUDIO firmware package.

Contents

List of tables

List of figures

1 Module overview

1.1 Algorithm functionality

The Graphical equalizer (GrEq) module is in charge of fine tuning the sound spectrum according to the 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 -12 dB to +12 dB in standard mode. The library can be generated with +18 dB or +24 dB maximum gains on request.

The current implementation uses 32-bit resolution for all computations, and can be used with both 16- and 32-bit input/output format at a sampling frequency of 48 kHz.

1.2 Module configuration

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

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

- *GrEq_CM4_IAR.a / GrEq_CM4_GCC.a / GrEq_CM4_Keil.lib*: for 16-bit input/output buffers, and running on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set
- *GrEq_32b_CM4_IAR.a / GrEq_32b_CM4_GCC.a / GrEq_32b_CM4_Keil.lib*: for 32-bit input/output buffers, and running on any STM32 microcontroller featuring an Arm® core with Cortex®-M4 instruction set
- *GrEq_CM7_IAR.a / GrEq_CM7_GCC.a / GrEq_CM7_Keil.lib*: for 16-bit input/output buffers, and running on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set
- *GrEq_32b_CM7_IAR.a / GrEq_32b_CM7_GCC.a / GrEq_32b_CM7_Keil.lib*: for 32-bit input/output buffers, and running on any STM32 microcontroller featuring an Arm® core with Cortex®-M7 instruction set

arm

1.3 Resources summary

Table 1 lists the requirements for memories and frequency. The footprints are measured on board, using IAR Embedded Workbench® for Arm® v7.40 (IAR Embedded Workbench® common components v7.2).

Table 1. Summary of resources

Use case (48 kHz, stereo interleaved)		Cortex® core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM(1)	Frequency (MHz)
Standard	5 bands	M4	4344 Bytes	108 Bytes	92 Bytes	548 Bytes	3840 Bytes	7.8
		M7(2)	4400 Bytes					6.1
	8 bands	M4	4344 Bytes					12.0
		M7(2)	4400 Bytes					9.3
	10 bands	M4	4344 Bytes					14.5
		M7(2)	4400 Bytes					11.5
Standard 32-bit I/O	5 bands	M4	4356 Bytes	108 Bytes	92 Bytes	548 Bytes	3840 Bytes	7.2
		M7(2)	4412 Bytes					5.4
	8 bands	M4	4356 Bytes					11.4
		M7(2)	4412 Bytes					8.5
	10 bands	M4	4356 Bytes					14.0
		M7(2)	4412 Bytes					10.5

1. Scratch RAM is the memory that can be shared with other processes running on the same priority level. This memory is not used from one frame to another by GrEq routines.
2. Footprints on STM32F7 Series MCUs (based on Cortex® M7 core) are measured on boards with stack and heap sections located in DTCM memory, with a 10 ms framing.

2 Module interfaces

Two files are needed to integrate the GrEq module. *GrEq_xxx_CMy_zzz.a/.lib* library and the *greq_glo.h* header file, which contains all definitions and structures to be exported to the software integration framework.

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:

- greq_reset function
- greq_setParam function
- greq_getParam function
- greq_setConfig function
- greq_getConfig function
- greq_process function

Each of these functions is described in the following sections.

2.1.1 greq_reset function

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

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

Table 2. Parameters for function greq_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 yet started.

2.1.2 greq_setParam function

This procedure writes the module static parameters from the main framework to the module internal memory. It can be called after reset routine and before real time processing starts. 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
*persistent_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	*persistent_mem_ptr*	*void **	Pointer to internal persistent memory
Returned value	-	*int32_t*	Error value

2.1.3 greq_getParam function

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

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

```
int32_t greq_getParam(greq_static_param_t *input_static_param_ptr, void
*persistent_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	*persistent_mem_ptr*	*void **	Pointer to internal persistent memory
Returned value	-	*int32_t*	Error value

2.1.4 greq_setConfig function

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
*persistent_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	*persistent_mem_ptr*	*void **	Pointer to internal persistent memory
Returned value	-	*int32_t*	Error value

2.1.5 greq_getConfig function

This procedure gets the module dynamic parameters from internal persistent 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
*persistent_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	*persistent_mem_ptr*	*void **	Pointer to internal persistent memory
Returned value	-	*int32_t*	Error value

2.1.6 greq_process function

This procedure is the module 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
*persistent_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	*persistent_mem_ptr*	*void **	Pointer to internal persistent 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, like the one 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: 2 for stereo
nb_bytes_per_Sample	*int32_t*	Dynamic of data in number of bytes: – 16 bits = 2 – 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: – 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 neither 16-bit nor 32-bit samples format
GREQ_UNSUPPORTED_NB_CHANNELS	-2	Input data is not 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: disables the effect, output buffer is equal to input buffer – 1: enables 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. The values are specified in dB, and go from -12 to +12 dB in 1 dB steps. 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. – 0: no preset, use the gains in the “user_gain_per_band_dB[]” table – 1: Pop – 2: Jazz – 3: Rock – 4: Vocal – 5: Classical – 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 that reduces drastically the neighboring effect of a band on the others. Each band gain can thus be set almost independently from the others. Each BPF is implemented using an optimized bi-quadratic structure, using 32-bit coefficients.

3.2 Data formats

Input of GrEq module is expected to be an audio stream, stereo/interleaved, in 16 or 32-bit format at 48 kHz sampling frequency. All operations are done with 32-bit resolution. The output format is an audio stream, stereo/interleaved signal in 16 or 32-bit 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

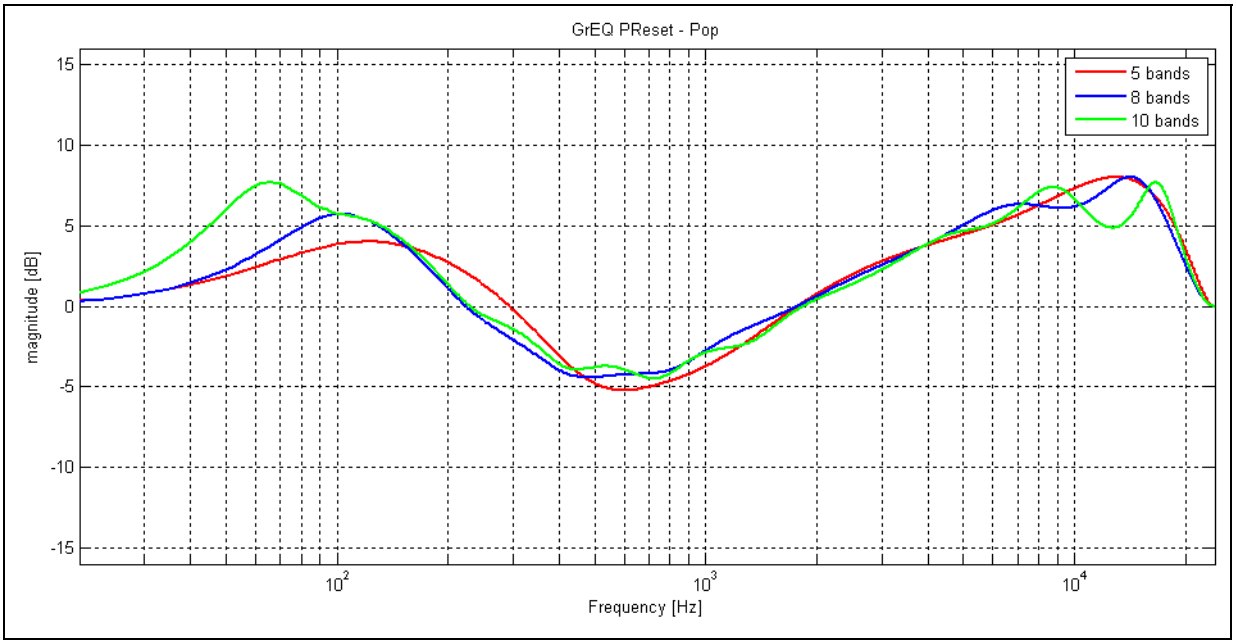


Figure 2. Frequency response for Jazz music

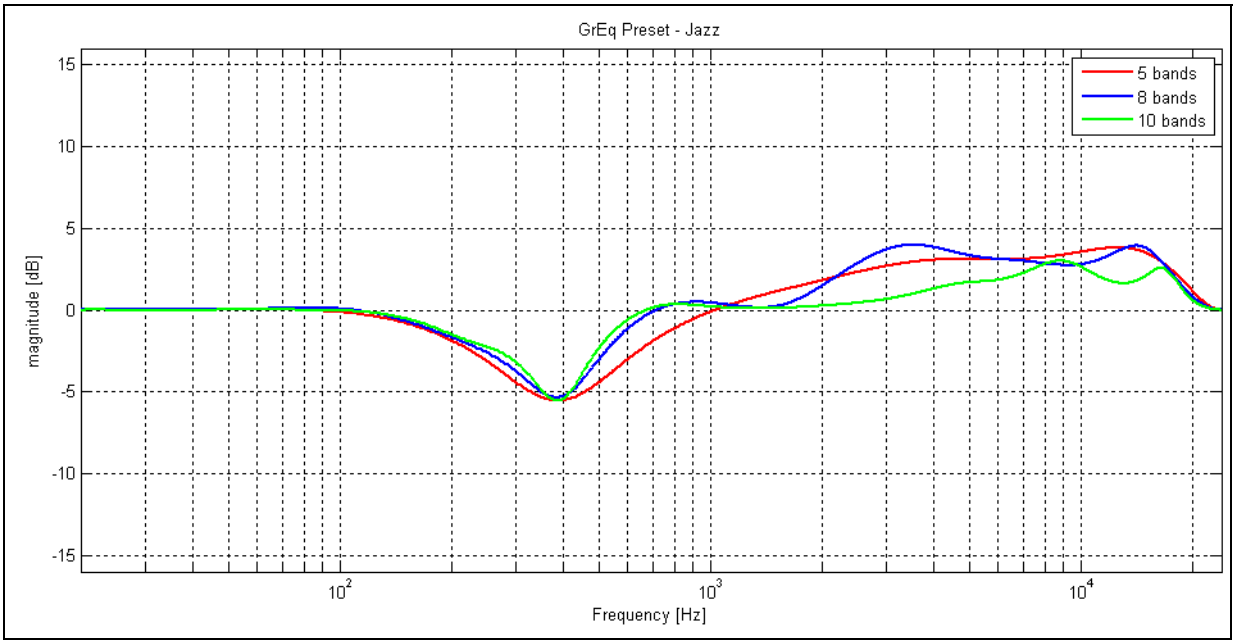


Figure 3. Frequency response for Rock music

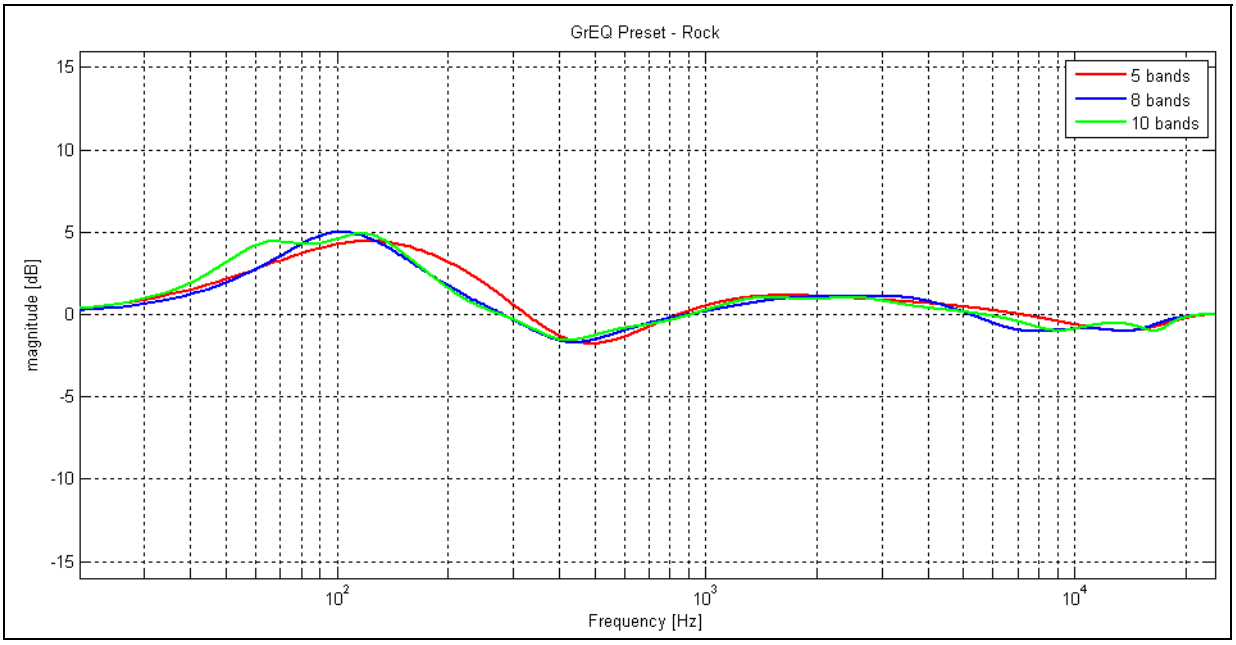


Figure 4. Frequency response for Vocal music

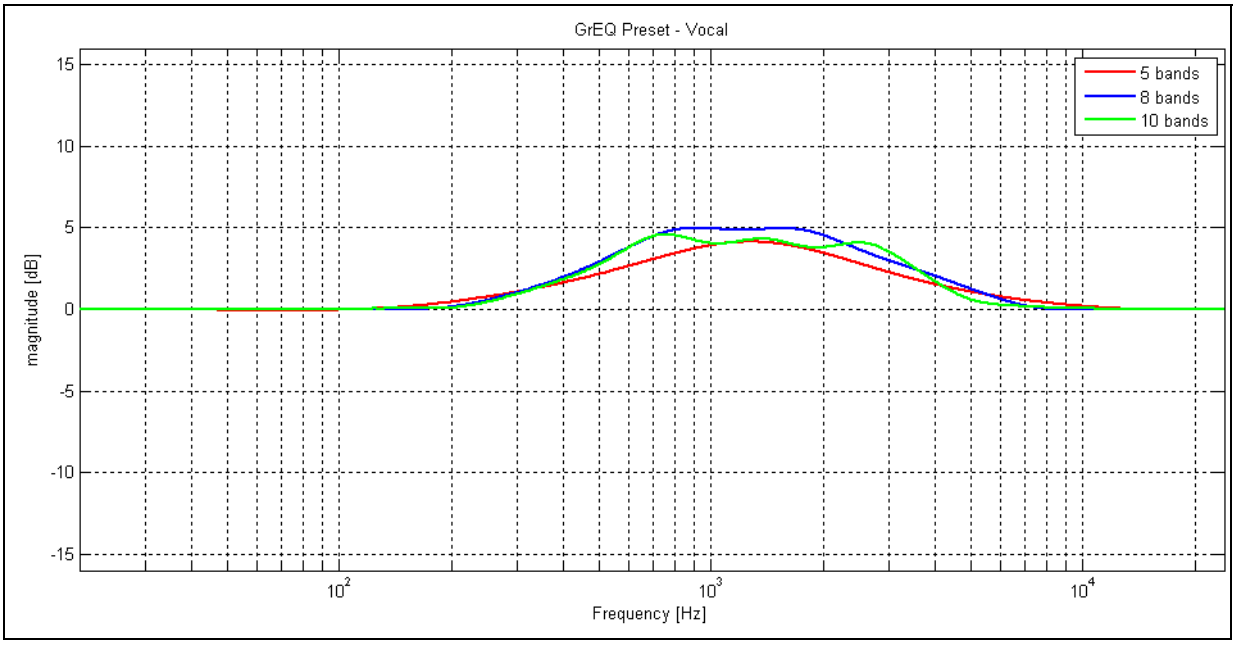


Figure 5. Frequency response for Classical music

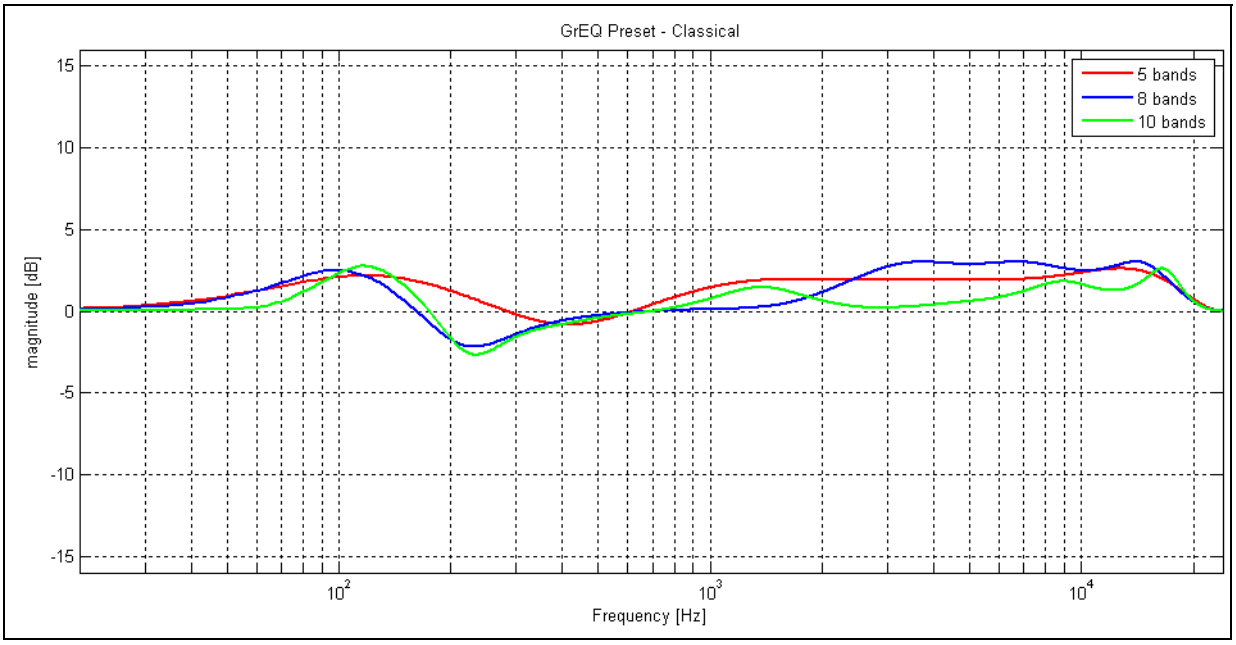
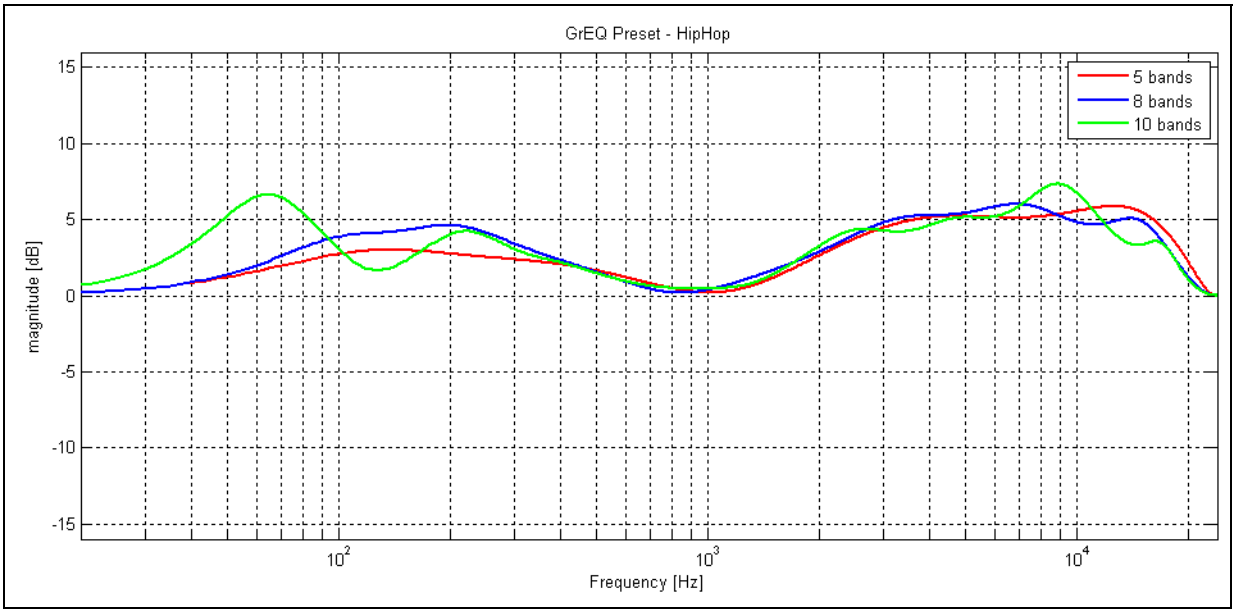


Figure 6. Frequency response for Hip Hop music



4 System requirements and hardware setup

GrEq libraries are built to run either on a Cortex®-M4 or on a Cortex®-M7 core without FPU usage, they can be integrated and run on microcontrollers of the STM32F4/STM32L4 or STM32F7 Series, respectively.

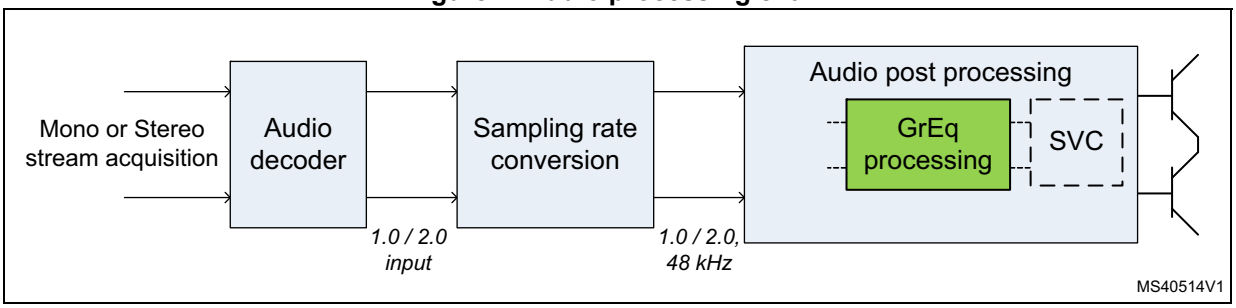
4.1 Recommendations for optimal setup

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 (see *Figure 7*), as the GrEq module (highlighted in green) applies pre-attenuation. Some margin is taken at its input, in order to avoid any saturation when the user is setting positive gains. The default available library is generated with a maximum gain of 12 dB, as a consequence two bits of guard are taken.

Another module, e.g. Smart Volume Control (SVC), can be used at the end of the audio chain to add these removed 12 dB without saturating. User manual UM1642, available on *www.st.com,* provides more information on the SVC module.

Figure 7. Audio processing chain



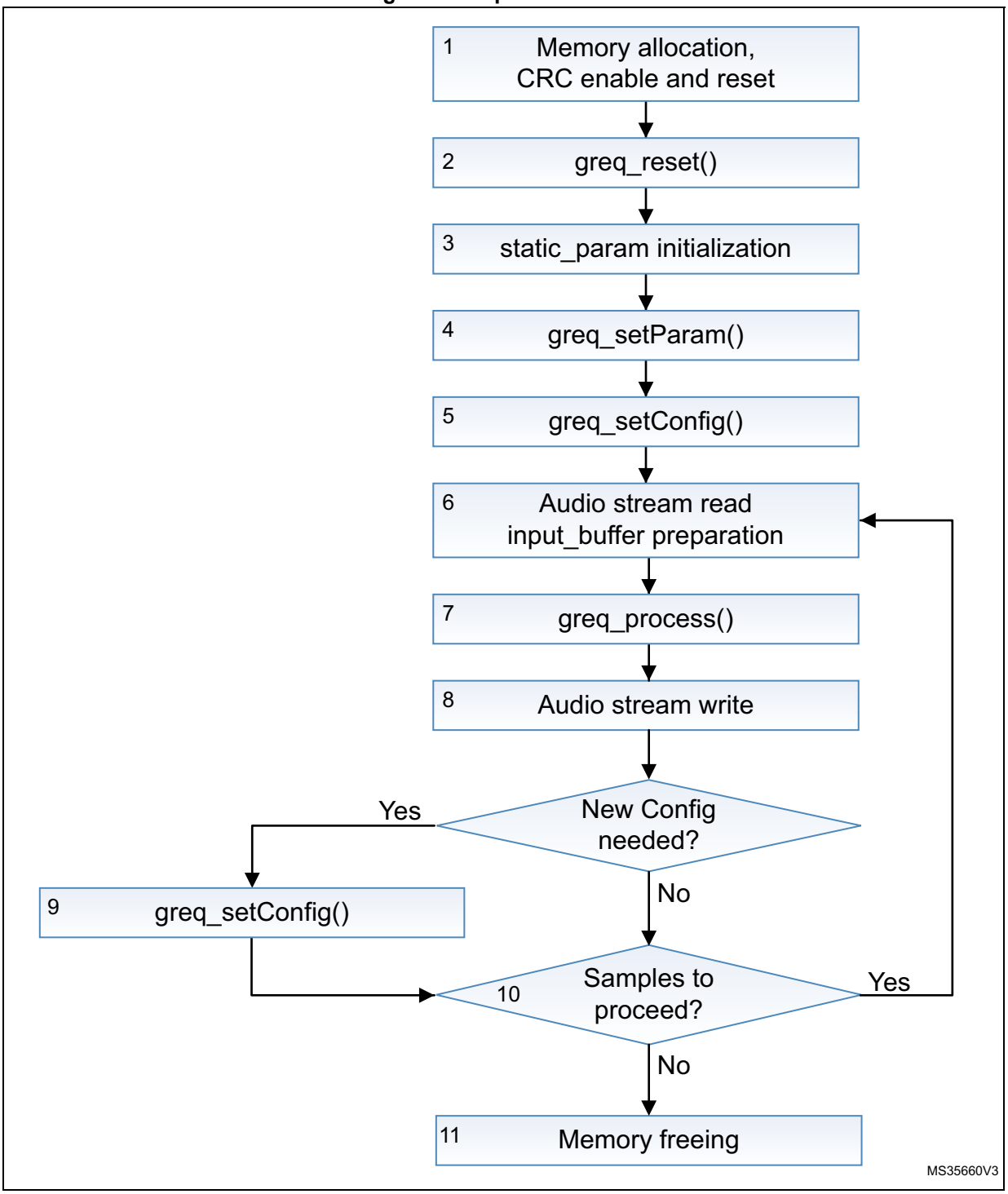
4.1.1 Module integration example

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

4.1.2 Module integration summary

The sequence is shown in *Figure 8*.

Figure 8. Sequence of APIs



1. As explained above, GrEq scratch and persistent memories have to be allocated, as well as input and output buffer according to the structures defined in *Section 2.2.1*. Furthermore, as GrEq library runs on STM32 devices, CRC HW block must be enabled and reset.
2. Once memory is 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.

5 How to tune and run the application

Once the module is integrated into an audio framework to play stereo samples at 48 kHz, launch a player and the output file will be decoded and played with audio spectrum modified following band gains of the graphical equalizer without returning any error message.

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.
04-Jan-2016	2	Scope extended to STM32Cube, hence updated document title and *Introduction*. Updated *Section 1.1: Algorithm functionality*, *Section 1.2: Module configuration*, *Section 1.3: Resources summary*, *Section 2.1: APIs*, *Section 3.2: Data formats*, *Section 4: System requirements and hardware setup* and *Section 5: How to tune and run the application*. Updated *Table 1: Summary of resources* and *Table 9: Error values*. Updated *Figure 8: Sequence of APIs*. Added *Section 4.1.1: Module integration example*. Added *Figure 7: Audio processing chain*. Removed former *Section 4.1.1: Memory allocation*.
22-Jan-2016	3	Updated *Section 5: How to tune and run the application*.
14-Mar-2017	4	Updated *Section 1.2: Module configuration*, *Section 1.3: Resources summary*, *Section 2: Module interfaces*, *Section 2.1.1: greq_reset function*, *Section 2.1.2: greq_setParam function*, *Section 2.1.3: greq_getParam function*, *Section 2.1.4: greq_setConfig function*, *Section 2.1.5: greq_getConfig function*, *Section 2.1.6: greq_process function*, *Section 4.1.1: Module integration example*, *Section 4.1.2: Module integration summary* and *Section 5: How to tune and run the application*. Updated *Table 1: Summary of resources*, *Table 2: Parameters for function greq_reset*, *Table 3: Parameters for function greq_setParam*, *Table 4: Parameters for function greq_getParam*, *Table 5: Parameters for function greq_setConfig*, *Table 6: Parameters for function greq_getConfig*, *Table 7: Parameters for function greq_process* and *Table 8: Input and output buffers*.
09-Jan-2018	5	Updated *Introduction* and *Section 1.2: Module configuration*.