UM2006
User manual

Gain Manager library software expansion for STM32Cube

Introduction

The Gain Manager (GAM) library user manual describes the software interface and requirements for the integration of the module into a main program such as the Audio STM32Cube expansion software and provides a basic understanding of the underlying algorithm.

The GAM library is used to apply several gain attenuations depending on channel number of audio signal.

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

Contents

List of tables

List of figures

1 Module overview

1.1 Algorithm function

The GAM module allows the user to modify digitally the volume of the input signal in the [-80;0] dB range, with a granularity of 0.5 dB.

1.2 Module configuration

The GAM module can handle mono, stereo and multichannel interleaved (up to eight channels) 16-bit and 32-bit I/O data.

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

- GAM_CM4_IAR.a / GAM_CM4_GCC.a / GAM_CM4_Keil.lib:
  16-bit input/output buffers version. It runs on any STM32 microcontroller featuring a core with a Cortex®-M4 core.
- GAM_32b_CM4_IAR.a / GAM_32b_CM4_GCC.a / GAM_32b_CM4_Keil.lib:
  32-bit input/output buffers version. It runs on any STM32 microcontroller featuring a core with a Cortex®-M4 core.
- GAM_CM7_IAR.a / GAM_CM7_GCC.a / GAM_CM7_Keil.lib:
  16-bit input/output buffers version. It runs on any STM32 microcontroller featuring a core with Cortex®-M7 core.
- GAM_32b_CM7_IAR.a / GAM_32b_CM7_GCC.a / GAM_32b_CM7_Keil.lib:
  32-bit input/output buffers version. It runs on any STM32 microcontroller featuring a core with Cortex®-M7 core.

1.3 Resources summary

Table 1 contains Flash, stack, RAM memory, and frequency (MHz) requirements of the module.Those footprints are measured on board, using IAR™ Embedded Workbench for ARM v7.40 (IAR™ Embedded Workbench common components v7.2).

Table 1. Resources summary(1)

Version	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM	Frequency (MHz)
16 bits I/O stereo	M4	1604 Bytes	8 Bytes	44 Bytes	120 Bytes	1 Byte	2.4
	M7	**1580 Bytes**					**1.3**
16 bits I/O multichannel 3.1	M4	1604 Bytes					4.4
	M7	**1580 Bytes**					**2.4**
32 bits I/O stereo	M4	1604 Bytes					2.6
	M7	**1580 Bytes**					**1.7**
32 bits I/O multichannel 3.1	M4	1604 Bytes					4.6
	M7	**1580 Bytes**					**3.0**

1. Data for M7 core are in bold type when different from those of M4.

Note: Footprints on STM32F7 are measured on boards with stack and heap sections located in DTCM memory.

Note: Scratch RAM is the memory that can be shared with other process running on the same priority level. This memory is not used from one frame to another by GAM routines.

2 Module Interfaces

Two files are needed to integrate the GAM module. *GAM_xxx_CMy_zzz.a/.lib* library and the gam_glo.h header file which contain all definitions and structures to be exported to the software integration 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:

- gam_reset
- gam_setParam
- gam_getParam
- gam_setConfig
- gam_getConfig
- gam_process

Each of these functions is described in the following sections.

2.1.1 gam_reset function

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

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

Table 2. gam_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 gam_setParam function

This procedure writes module’s 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 parameters with the values which cannot be changed during the module processing.

```
int32_t gam_setParam(gam_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 3. gam_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	gam_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 gam_getParam function

This procedure gets the module's 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 gam_getParam(gam_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 4. gam_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	gam_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 gam_setConfig function

This procedure sets the module's dynamic parameters from the main framework to the module internal memory. It can be called at any time during the module processing (after the reset and setParam routines).

```
int32_t gam_setConfig(gam_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. gam_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	gam_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 gam_getConfig function

This procedure gets the module's dynamic parameters from the internal persistent memory to the main framework. It can be called at any time during processing (after the reset and setParam routines).

```
int32_t gam_getConfig(gam_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 6. gam_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	gam_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 gam_process function

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

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

Table 7. gam_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
Returned value	–	int32_t	Error value

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

2.2 External definitions and types

Some types and definitions have been defined to facilitate the integration in the main frameworks.

2.2.1 Input and output buffers

The GAM 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 and filled in by the main framework before each call to the processing routine:

```
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 or 2 for stereo for instance
nb_bytes_per_Sample	int32_t	Dynamic data in number of bytes (2 for 16-bit data, 4 for 32-bit)
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	Buffer mode: 0 = not interleaved, 1 = interleaved

2.2.2 Returned error values

Possible returned error values are described below:

Table 9. Returned error values

Definition	Value	Description
GAM_ERROR_NONE	0	OK - No error detected
GAM_UNSUPPORTED_VOLUME	-1	Volume setting is outside [-80; 0] db range
GAM_UNSUPPORTED_MUTE_MODE	-2	Multi mode is not supported
GAM_UNSUPPORTED_NUMBER_OF_BYTEPERSAMPLE	-3	Input data is neither 16-bit nor 32-bit value
GAM_UNSUPPORTED_INTERLEAVING	-4	Input data is stereo / not interleaved
GAM_UNSUPPORTED_MULTICHANNEL	-5	Number of channels is not supported
GAM_BAD_HW	-6	The library is not used with the right hardware

2.3 Static parameters structure

There is no static parameter to be used. For the compatibility with other structures, the static parameter structure contains a dummy field.

```
struct gam_static_param {
    int8_t   empty;
};
typedef struct gam_static_param gam_static_param_t;
```

Table 10. Static parameters structure

Name	Type	Description
empty	int8_t	Dummy field, just required to have a non-empty structure

2.4 Dynamic parameters structure

Three dynamic parameters can be used.

```
struct gam_dynamic_param {
    int16_t   target_volume_dB[8];
    int16_t   mute[8];
};
typedef struct gam_dynamic_param gam_dynamic_param_t;
```

Table 11. Dynamic parameters structure

Name	Type	Description
target_volume_dB[8]	int16_t	Volume dB input value, in 1/2 dB steps, for each of the 8 different channels. Example:” -12,-12,-12,-12,-13,-13,-13,-13” means the first 4 channels will get a target output volume of -6 dB, the 4 others a target volume of -6.5 dB
mute[8]	int16_t	1 = enable mute, 0 = disable, for each of the 8 different channels.

3 Algorithm description

3.1 Processing steps

The GAM algorithm is composed of one main processing block (called gain application block), that attenuates the signal, depending on the target volume value set among the dynamic parameters.

It can be different for each channel.

Every target volume value change is applied with smooth transition to the output signal.

3.2 Data formats

The GAM module supports fixed point mono and stereo multichannel interleaved 16-bits and 32-bit input data. It can work independently of the frame size and the input signal sampling rate.

However it is recommended to avoid selecting very short frame size (i.e. lower than 2 ms), in order to prevent “plops” on transitions due to too short ramp up/down of the volume.

The module delivers output data, following the same interleaved pattern and 16-bit or 32-bit resolution, as the input data.

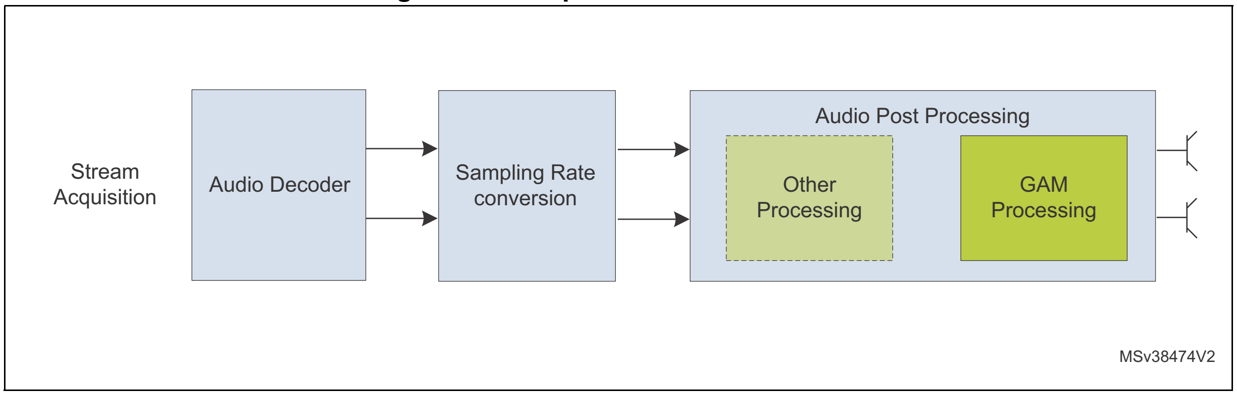
4 System requirements and hardware setup

GAM 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 corresponding STM32F4/STM32L4 or STM32F7 series devices. There is no other hardware dependency.

4.1 Recommendations for an optimal setup

It is recommended to execute the GAM library close to the audio DAC to avoid decreasing the Signal to Noise Ratio (SNR) before other post processing modules execution. Refer to *Figure 1: Example of basic audio chain*

Figure 1. Example of basic audio chain

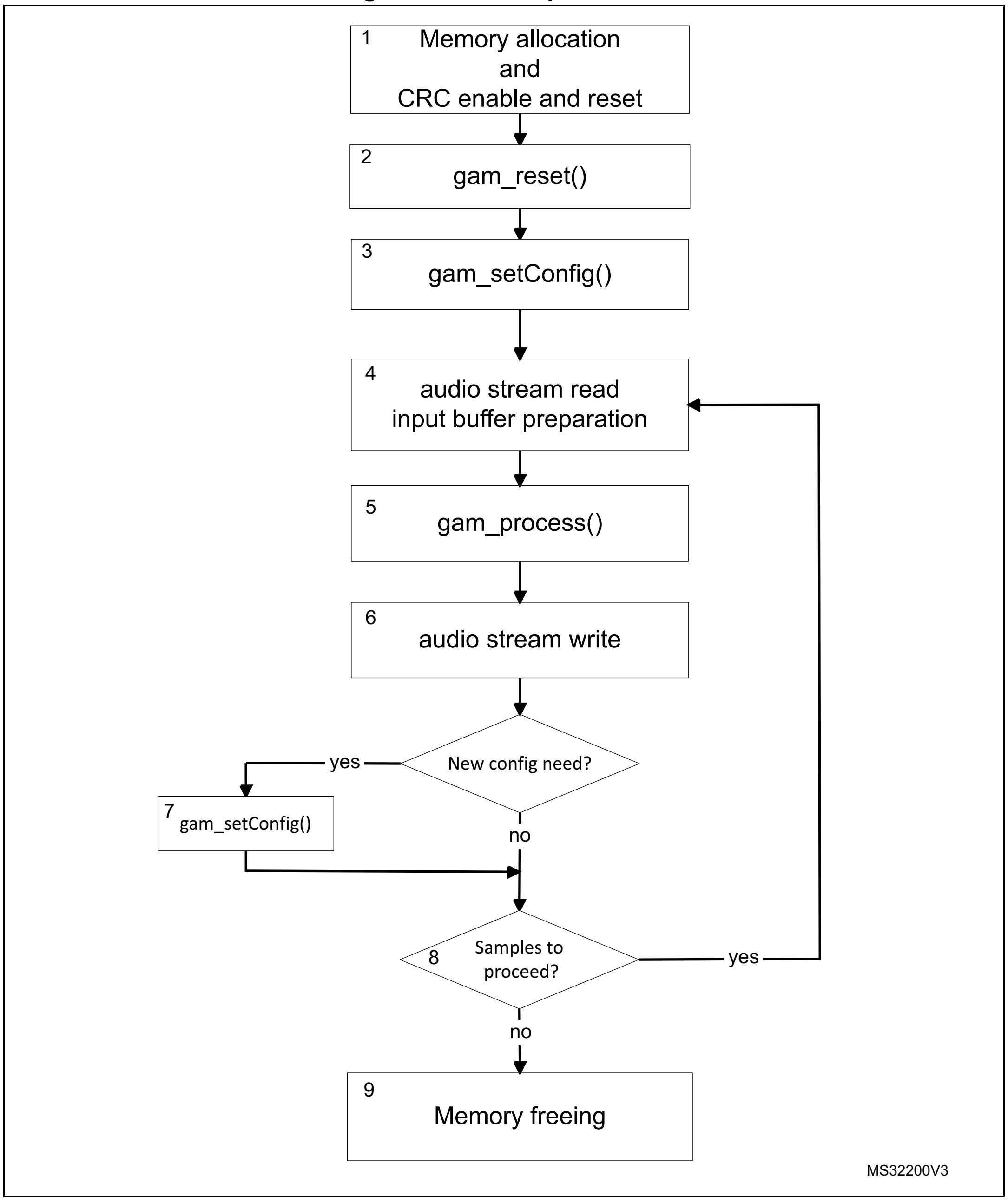


4.1.1 Module integration example

Cube expansion GAM integration examples are provided on 32F746GDISCOVERY and 32F469IDISCOVERY boards. Please refer to provided integration code for more details.

4.1.2 Module integration summary

Figure 2. API call procedure



1. As explained above, GAM static and dynamic structures have to be allocated, as well as input and output buffers. Furthermore, as GAM library runs on STM32 devices, CRC HW block must be enable and reset.
2. Once the memory is allocated, the call to gam_reset() function initializes the internal variables.
3. Once the dynamic parameters are updated, the gam_setConfig() routine is called to send the dynamic parameters from the audio framework to the module.
4. The audio stream is read from the proper interface and input_buffer structure has to be filled accordingly to the stream characteristics (number of channels, sample rate, interleaving and data pointers). Output buffer structure has to be set as well.
5. Main processing routine is called to apply the effect.
6. The output audio stream can now be written in the proper interface.

7. If needed, the user can set new dynamic parameters and call the gam_setConfig() routine again, to update the module configuration.
8. If the application is still running and has new input samples to proceed, then it goes back to step 4, else the processing loop is over.
9. Once the processing loop is over, the allocated memory has to be freed.

5 How to run and tune the application

Once the module is integrated into the audio framework to play samples at 48 kHz, launch the Audio player and, if there is no sampling rate conversion available, choose a .wav or .mp3 file with a 48 kHz sampling frequency.

The GAM target gain can be controlled through the user interface in the provided integration examples.

6 Revision history

Table 12. Document revision history

Date	Revision	Changes
20-Jan-2016	1	Initial release.
08-Jan-2018	2	Replaced *static memory* with *persistent memory*, and *dynamic memory* with *scratch memory* throughout the document. Changed parameter *static_mem_ptr* to *persistent_mem_prt*, and *dynamic_mem_ptr* parameter to *scratch_mem_prt*. Updated *Section 1.2: Module configuration* and *Section 1.3: Resources summary*. Updated *Table 8: Input and output buffers* for channel numbers. Simplified *Section 4.1.1: Module integration example* and *Chapter 5: How to run and tune the application*. Updated discovery kit references in *Section 4.1.1*. Replace X-CUBE-AUDIO-F4, X-CUBE-AUDIO-F7 and X-CUBE-AUDIO-L4 with X-CUBE-AUDIO.