# UM1778
# User manual

## BAss Manager library software expansion for STM32Cube

## Introduction

This user manual describes the software interface and the requirements of the BAss Manager (BAM) module. It provides details on all the necessary interface functions, the parameters and the constraints to integrate the BAM library into a program like the Audio STM32Cube expansion software.

The BAM library is part of the following firmware packages

- X-CUBE-AUDIO-F4
- X-CUBE-AUDIO-L4
- X-CUBE-AUDIO-F7

# Contents

# List of tables

# List of figures

# 1 Module overview

## 1.1 Algorithm functionality

The BAss Manager (BAM) module has the function of modifying the low-frequency part of a stereo signal.

It is possible to configure the cross-over cut-off frequency and to amplify or attenuate the bass signal. The modified low-frequency content can be mixed back with the rest of the signal or outputted as a third signal.

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

## 1.2 Module configuration

BAM module supports Stereo interleaved 16 or 32 bits 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. The minimum input frame is 96 stereo samples, corresponding to 2 ms scheduling at 48 kHz sampling frequency.

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

- *BAM_CM4_IAR.a / BAM_CM4_GCC.a / BAM_CM4_Keil.lib*: BAM library with 16 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set
- *BAM_32b_CM4_IAR.a / BAM_32b_CM4_GCC.a / BAM_32b_CM4_Keil.lib*: BAM library with 32 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set
- *BAM_CM7_IAR.a / BAM_CM7_GCC.a / BAM_CM7_Keil.lib*: BAM library with 16 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set
- *BAM_32b_CM7_IAR.a / BAM_32b_CM7_GCC.a / BAM_32b_CM7_Keil.lib*: BAM library with 32 bits input/output buffers, running on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set

## 1.3 Resources summary

*Table 1* summarizes CPU, Flash memory, Stack and RAM requirements. The core MHz value is an average measured on real hardware. The footprints are measured on board, using IAR Embedded Workbench® for ARM® v7.40 (IAR Embedded Workbench® common components v7.2).

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

**Table 1. Resources summary**

| I/O | Core | BAM mode (48 kHz) | Flash code (.text) | Flash data (.rodata) | Stack | Persistent RAM | Scratch RAM | Limiter | |
|---|---|---|---|---|---|---|---|---|---|
| | | | | | | | | ON | OFF |
| | | | Bytes | | | | | MHz | |
| 16 bits | M4 | BAM_OUTPUT_MODE_2_0 | 7694 | 1816 | 140 | 2236 | 5760 | 21.4 | 14.4 |
| | | BAM_OUTPUT_MODE_2_1 _WITH_LFE_SPLIT | | | | | | 17.8 | 14.4 |
| | | BAM_OUTPUT_MODE_2_1 _WITHOUT_LFE_SPLIT | | | | | | 14.7 | 11.2 |
| | M7(1) | BAM_OUTPUT_MODE_2_0 | 7726 | | | | | 16.1 | 9.9 |
| | | BAM_OUTPUT_MODE_2_1 _WITH_LFE_SPLIT | | | | | | 13.0 | 9.9 |
| | | BAM_OUTPUT_MODE_2_1 _WITHOUT_LFE_SPLIT | | | | | | 10.9 | 7.8 |
| 32 bits | M4 | BAM_OUTPUT_MODE_2_0 | 7742 | | | | | 21.0 | 14.1 |
| | | BAM_OUTPUT_MODE_2_1 _WITH_LFE_SPLIT | | | | | | 17.8 | 14.4 |
| | | BAM_OUTPUT_MODE_2_1 _WITHOUT_LFE_SPLIT | | | | | | 14.8 | 11.3 |
| | M7(1) | BAM_OUTPUT_MODE_2_0 | 7726 | | | | | 15.8 | 9.7 |
| | | BAM_OUTPUT_MODE_2_1 _WITH_LFE_SPLIT | | | | | | 13.0 | 9.9 |
| | | BAM_OUTPUT_MODE_2_1 _WITHOUT_LFE_SPLIT | | | | | | 11.0 | 7.8 |

1. Footprints on STM32F7xx MCUs are measured on boards with stack and heap sections located in DTCM memory.

# 2 Module interfaces

Two files are needed to integrate the BAM module. *BAM_xxx_CMy_zzz.a/.lib* and *bam_glo.h* header file contain all definitions and structures to be exported to the application software.

Note 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:

- bam_reset
- bam_setParam
- bam_getParam
- bam_setConfig
- bam_getConfig
- bam_process

### 2.1.1 bam_reset function

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

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

**Table 2. Parameters for function bam_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 bam_setParam function

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

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

```
int32_t bam_setParam(bam_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 3. Parameters for function bam_setParam

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_static_param_ptr* | *bam_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 bam_getParam function

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 bam_getParam(bam_static_param_t
*input_static_param_ptr, void *persistent_mem_ptr);
```

Table 4. Parameters for function bam_getParam

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_static_param_ptr* | *bam_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 bam_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 bam_setConfig( bam_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. Parameters for function bam_setConfig

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_dynamic_param_ptr* | *bam_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 bam_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 bam_getConfig(bam_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

**Table 6. Parameters for function bam_getConfig**

| I/O | Name | Type | Description |
|---|---|---|---|
| **Input** | *input_dynamic_param_ptr* | *bam_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 bam_process function

This procedure is the module's main processing routine.

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

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

**Table 7. Parameters for function bam_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, meaning that the same buffer can be used for input and output for BAM_OUTPUT_MODE_2_0 mode.

## 2.2 External definitions and types

### 2.2.1 Input and output buffers

The BAM 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: 2 for stereo, 3 for 2.1 multichannel |
| *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:<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 |
|---|---|---|
| BAM_ERROR_NONE | 0 | No error |
| BAM_UNSUPPORTED_INTERLEAVING_MODE | -1 | Input data is stereo/not interleaved |
| BAM_UNSUPPORTED_NB_CHANNELS | -2 | Input data is not stereo |
| BAM_UNSUPPORTED_NB_OF_BYTEPERSAMPLE | -3 | Input data is neither 16-bit nor 32-bit sample format |
| BAM_UNSUPPORTED_XOVER_FREQUENCY | -4 | Cross-over frequency is not compatible |
| BAM_UNSUPPORTED_MODE | -5 | Unsupported output mode |
| BAM_BAD_HW | -6 | Unsupported HW for the library |
| BAM_UNSUPPORTED_FRAME_SIZE | -7 | The input frame size is either too big or too small |

## 2.3 Static parameters structure

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

```
struct bam_static_param {
    int16_t  mode;
}
```

A description of different available modes is presented in *Table 10*.

**Table 10. Modes and corresponding configurations**

| Name | Type | Description |
|---|---|---|
| *mode* | *int16_t* | 1 = output is 2.0 configuration.<br>Low frequencies are extracted, amplified or attenuated and mixed back to high frequencies. |
| | | 2 = output is 2.1 configuration.<br>Low frequencies are extracted, amplified or attenuated and outputted in a third channel (the x.1 part). Left and Right channel (the 2.y part) are kept with low frequencies filtered out. |
| | | 3 = output is 2.1 configuration.<br>Low frequencies are extracted, amplified or attenuated and outputted in a third channel (the x.1 part). Left and Right channel (the 2.y part) are kept not modified. |

## 2.4 Dynamic parameters structure

It is possible to change the BAM configuration by setting new values in the dynamic parameter structure (see *Table 11*) before calling the *bam_setConfig()* function.

```
struct bam_dynamic_param {
    int32_t  limiter_release_time;
    int16_t  bass_vol;
    int16_t  freq_xover;
    int16_t  enable;
    int16_t  limiter_enable;
}
```

**Table 11. Dynamic parameters structure**

| Name | Type | Description |
|---|---|---|
| *limiter_release_time* | *int32_t* | Release time for the limiter attenuation: the gain is smoothed through a first order filter.<br>The coefficient is equal to *limiter_release_time* parameter, in Q31 format: $limiter\_release\_time_{int32} = round(coeff_{dec} * 2^{31})$, with $coeff_{dec} = exp(-1/Rt*Fe)$, with Rt = release time (in second) and Fe = sampling rate (in Hz).<br>Default value for Rt is 200 ms, values between 1 and 900 ms make sense. |
| *bass_vol* | *int16_t* | Volume level for the bass signal in dB.<br>Supported values: form -24 to +24 with 1dB step. |

**Table 11. Dynamic parameters structure (continued)**

| Name | Type | Description |
|---|---|---|
| *freq_xover* | *int16_t* | Cut-off frequency, in Hz, of the cross-over filter which split the input signal into the bass signal and the medium/high frequencies spectrum.<br>Supported values: 60, 70, 80, 100, 120, 150, 180, 200, 220, 250, 280, 300. |
| *enable* | *int16_t* | 1 = enables the processing of the BAM module.<br>0 = pass-through configuration. |
| *limiter_enable* | *int16_t* | 1 = enable, output of BAM is processed through limiter.<br>0 = disable, no limiter is applied. |

# 3 Algorithm description

## 3.1 Processing steps

The BAM block diagram for output mode=1 is shown in *Figure 1*. The algorithm starts with the extraction of the low frequency part, based on biquadratic filter blocks.

A 6$^{th}$ order low-pass filter is used for low frequency extraction, this guarantees a high roll-off. The frequency cut-off is selectable from a various number of values. The low frequency signal is then processed by a non-linear volume processor (SVC), which can attenuate the signal or amplify it. In this last case, if the input single amplitude plus the bass_vol level exceeds the digital range, a non-linear processing (compressor) is applied.

The high frequency part is obtained by filtering the input signal with an high-pass filter, the cut-off frequency is the same of the low-pass filter.

When output mode is BAM_OUTPUT_MODE_2_0, the processed low frequency signal is mixed back to the high frequency signal. If necessary, a limiter can be enabled in order to soft-clip the signal after mixing. The limiter has a 20:1 ratio and a transition phase set so that output is limited at 0 dBFS.

**Figure 1. BAM block diagram for mode BAM_OUTPUT_MODE_2_0**

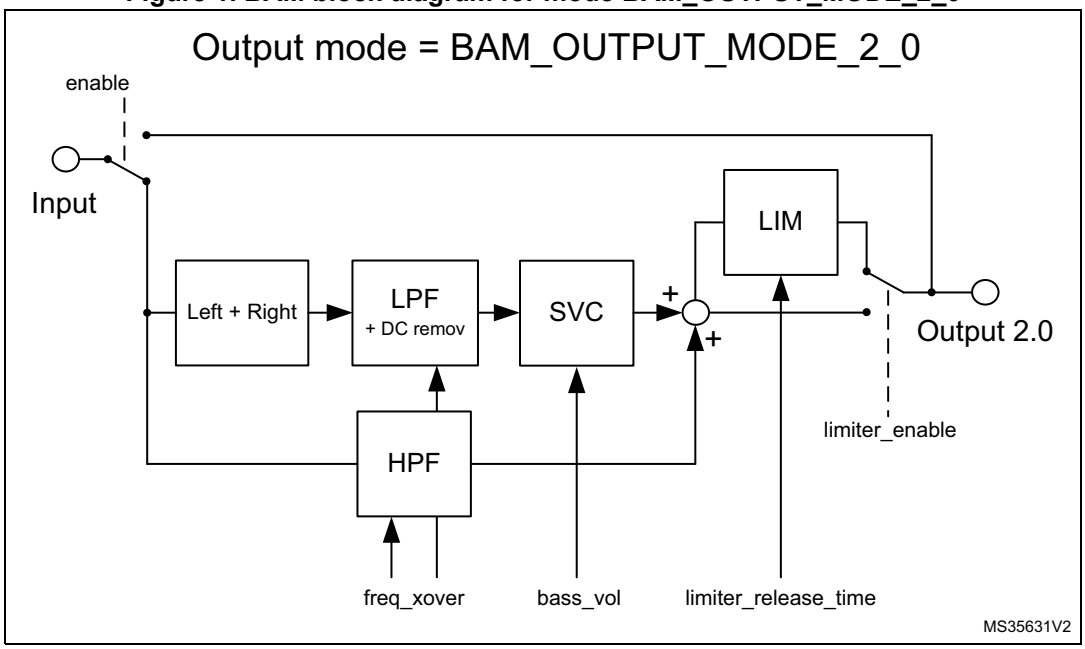



*Figure 2* and *Figure 3* detail, respectively, the block diagrams for output modes BAM_OUTPUT_MODE_2_1_WITH_LFE_SPLIT and BAM_OUTPUT_MODE_2_1_WITHOUT_LFE_SPLIT.

**Figure 2. BAM block diagram for mode BAM_OUTPUT_MODE_2_1_WITH_LFE_SPLIT**

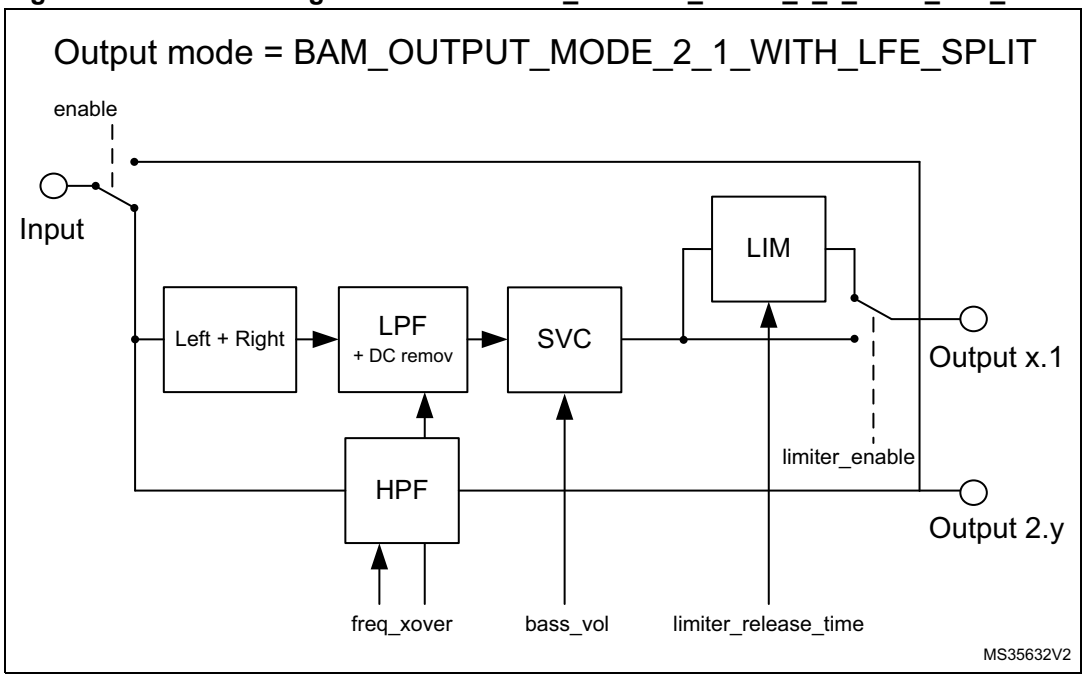
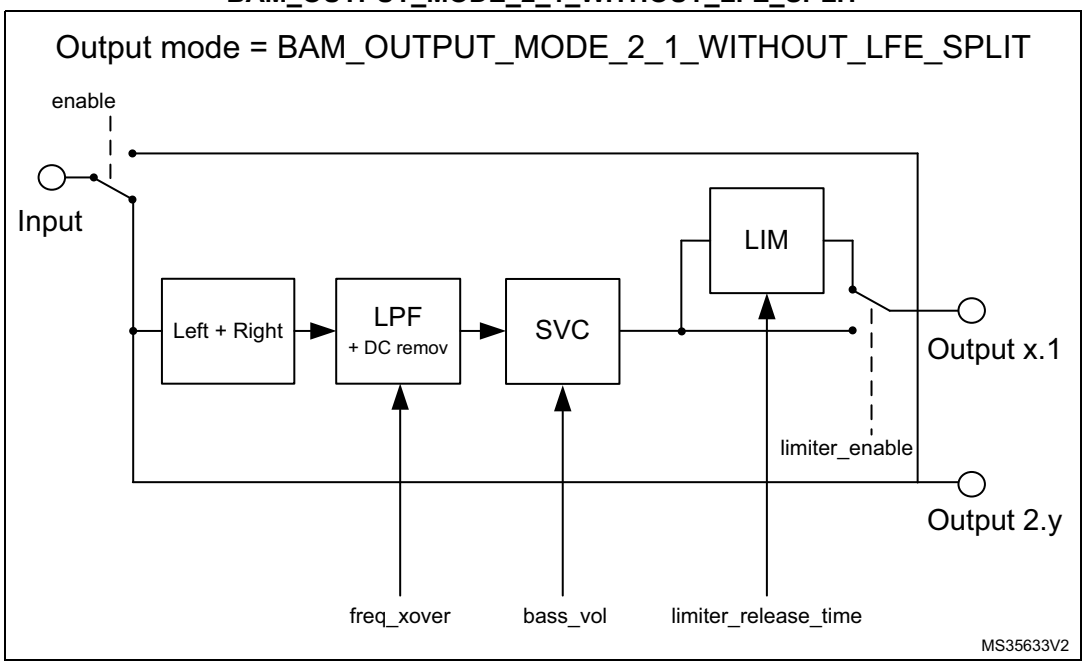



**Figure 3. BAM block diagram for mode BAM_OUTPUT_MODE_2_1_WITHOUT_LFE_SPLIT**




## 3.2 Data formats

Input of BAM module is expected to be an audio stream, stereo/interleaved, in 16 or 32 bits format. All operations are performed with 32 bits resolution.

The output format is depending on the “mode” parameter value. If mode is BAM_OUTPUT_MODE_2_0, then output is a stereo/interleaved signal in 16 or 32 bits format. If mode is BAM_OUTPUT_MODE_2_1_WITH_LFE_SPLIT or BAM_OUTPUT_MODE_2_1_WITHOUT_LFE_SPLIT, then output is a three channels interleaved signal in 16 or 32 bits format.

## 3.3 Performance measurements

### 3.3.1 Dynamic curves

*Figure 4* shows some characterization curves of the BAM algorithm, while three examples of compression curves for bass_vol=6, 12 and 24 dB are reported in *Figure 5*.

The compression curve of the limiter is sketched in *Figure 6*.

**Figure 4. Crossover filters example for Fc=80 Hz**

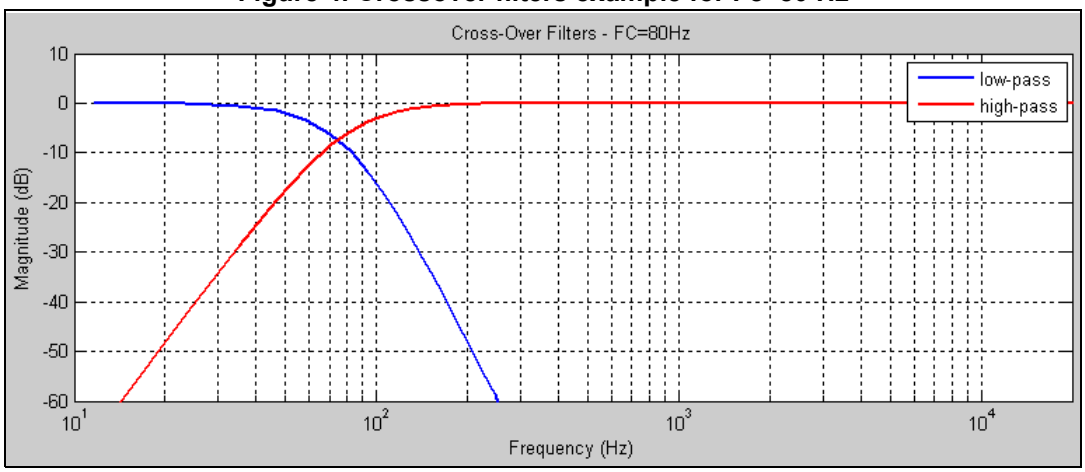


**Figure 5. Compression curve examples**

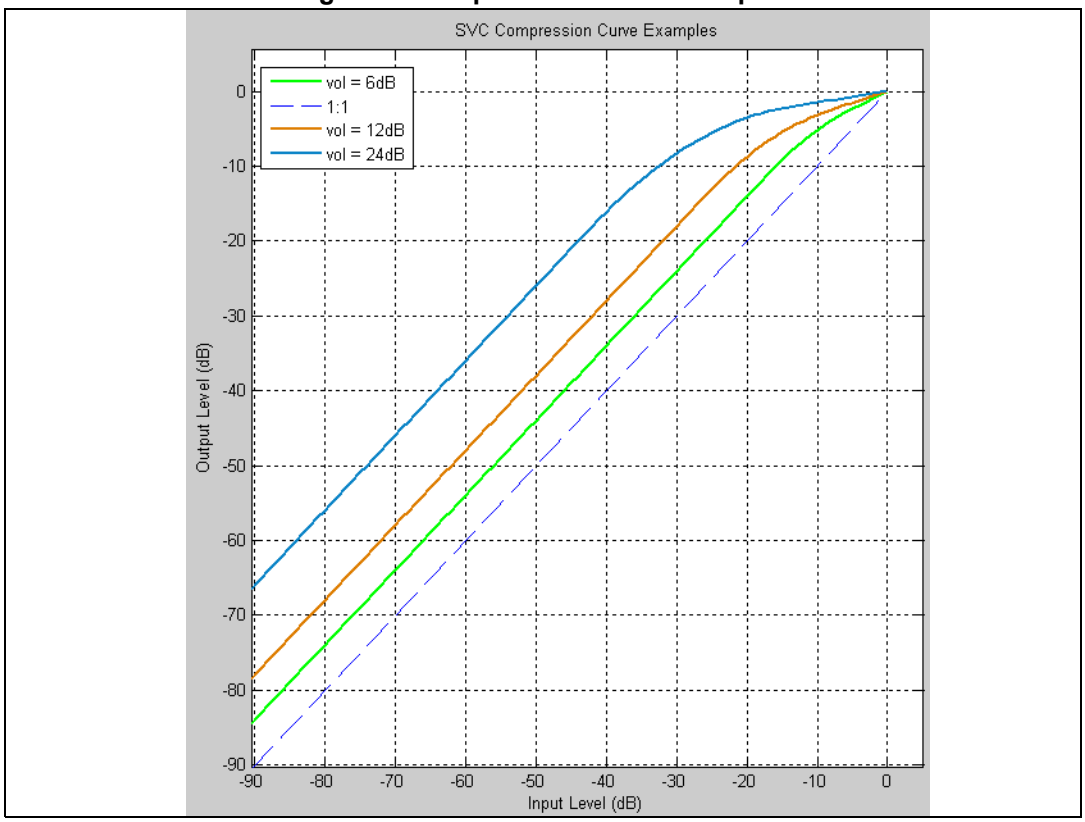
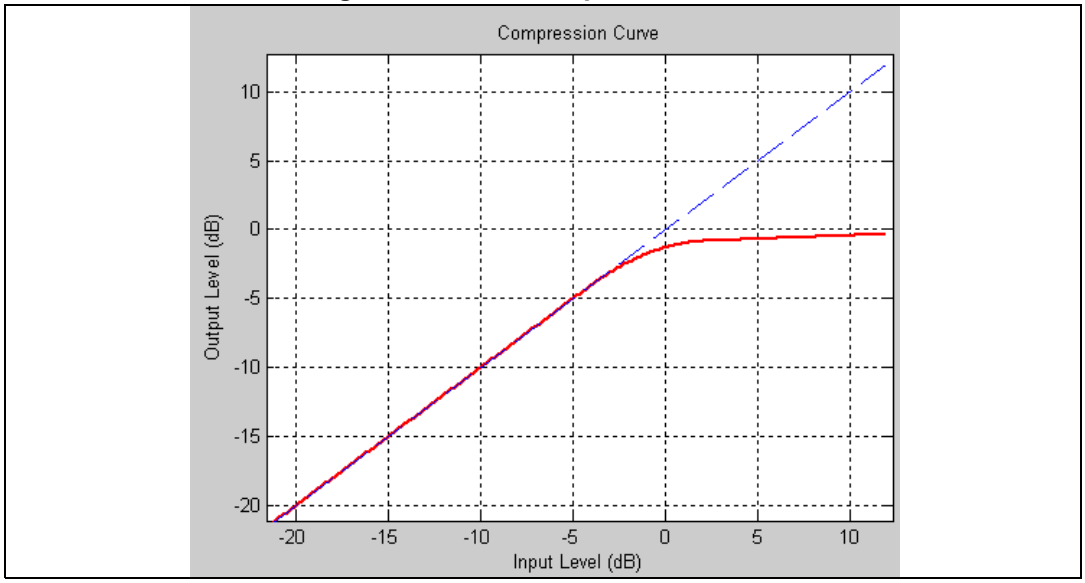
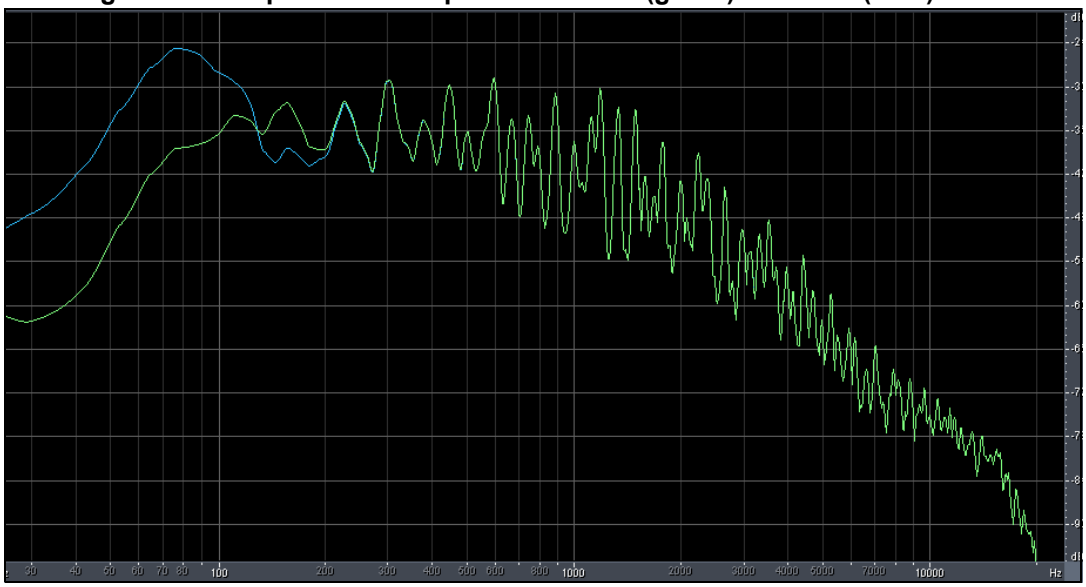



**Figure 6. Limiter compression curve**




In *Figure 7* “bass_vol” parameter is set to 18dB, “xover_freq” is set to 120 Hz.

Figure 7. Example of music spectrum before (green) and after (blue) BAM.



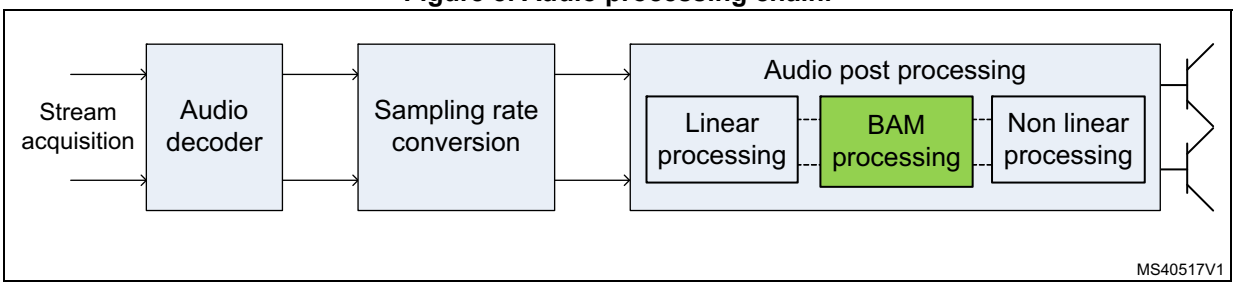
# 4 Application description

## 4.1 Recommendations for optimal setup

The BAM module can be executed at any place in an audio processing chain. Nevertheless, as it applies non-linear algorithm, it should be executed after linear algorithms (such as GrEQ, virtualization, filtering). It should be also executed before the Smart Volume Control (SVC) algorithm, that acts as a master volume.

*Figure 8* (where the BAM module is highlighted in green) shows the optimal setup.

**Figure 8. Audio processing chain.**




### 4.1.1 Module integration example

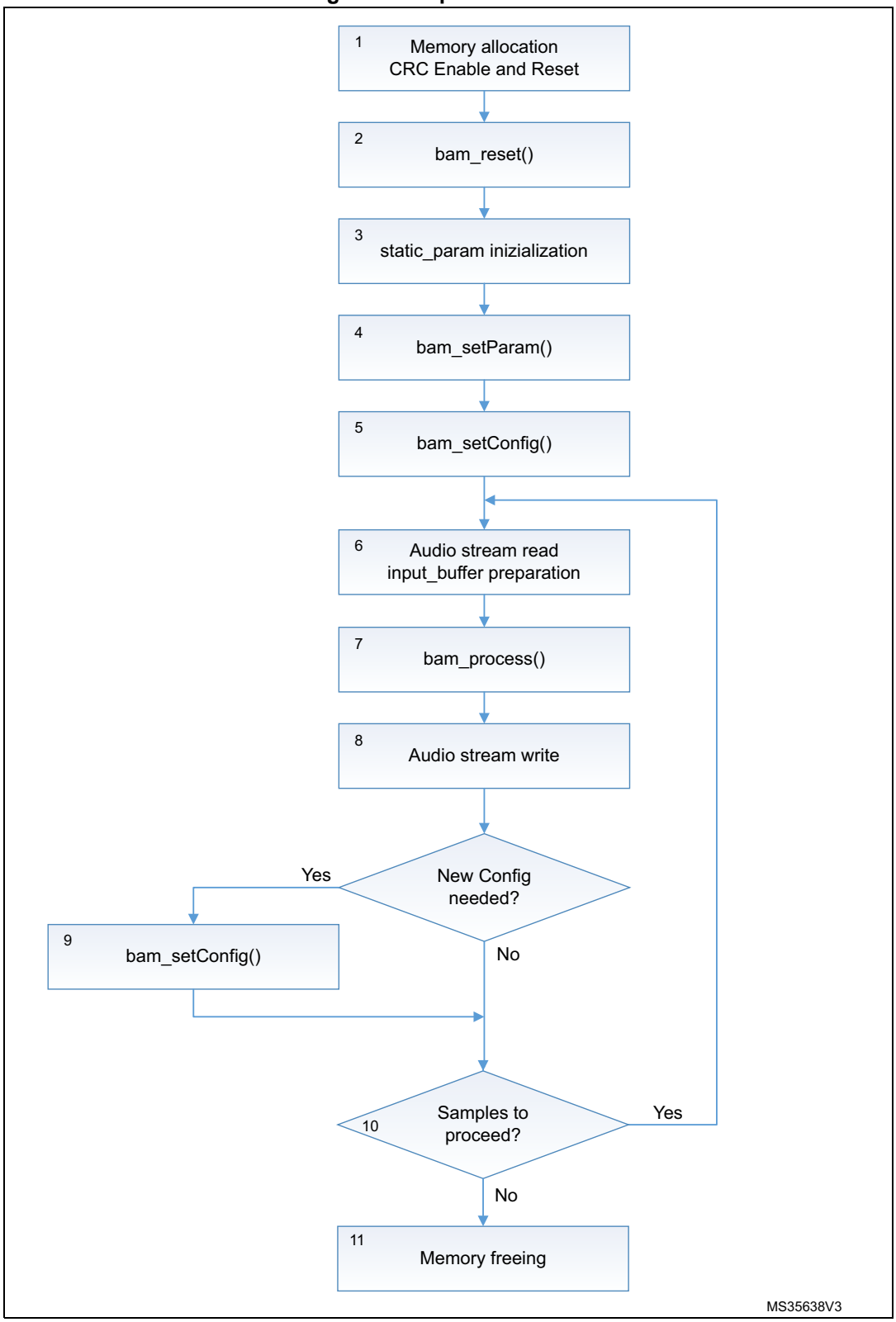
Cube expansion BAM integration examples are provided on STM32F746G-Discovery and STM32F469I-Discovery boards. Refer to the provided integration code for more details.

### 4.1.2 Module APIs calls

The sequence is shown in *Figure 9*, and each step is described in detail in the following list:

1. As explained above, BAM scratch and persistent memories have to be allocated, as well as input and output buffer accordingly to the structures defined in *Section 2.2.1*.
2. Once memory is allocated, the call to *bam_reset()* function will initialize internal variables.
3. The BAM mode parameters can now be set by initializing the static_param structure.
4. Call *bam_setParam()* routine to apply static parameters.
5. Alternatively, “enable”, “bass_vol”, “freq_xover”, “limiter_enable” and “limiter_release_time” can be changed by setting the dynamic parameters structure and calling *bam_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. Call to *bam_process()* function will execute the BAM 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 *bam_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 9. Sequence of APIs**



# 5 How to tune and run the application

User should refer to the provided integration framework running on STM32F469I-Discovery board for X-CUBE-AUDIO-F4 and STM32F746G-Discovery board for X-CUBE-AUDIO-F7. This example of BAM library integration is built using IAR systems® v7.40 tool chain.

Once the module has been integrated into an audio framework to play stereo samples at 48 kHz, user must launch a player and the output file will be decoded and played with BAM effect tunable.

The effect of the BAM algorithm will partly depend on the input signal. As a matter of fact, if the music is already strongly compressed, with a low crest-factor and that low frequencies are already at a high level, the BAM module will not be able to amplify the bass level so much, even with a *bass_vol* parameter set to high values. Nevertheless, as long as the input signal has some margin for bass signal to be amplified or compressed, BAM algorithm will be able to do so.

The value for the cross-over frequency will mainly depend on the speaker on which bass signal is played. Setting a “xover_freq” parameter to 60 Hz while the speaker has a cut-off frequency starting at 120 Hz will not be perceived by the user, simply because the speaker is mechanically unable to reproduce these low frequencies. In this case, “xover_freq” parameter should be increased.

On the other hand, setting *xover_freq* to high values (> 200 Hz) should be done with care because it will start to degrade the spectrum balance of the played music.

*Figure 10* and *Figure 11* are two different examples of input signals, processed with the same BAM settings. The crest-factor indicator used is the ratio of the peak value over the root-mean-square value of the signal. In both figures signal is normalized to full-scale.

The main parameters for these two examples are summarized in *Table 12*.

**Table 12. Parameters for two different music styles**

| | | Hiphop music | Pop music |
|---|---|---|---|
| Features | | Lot of bass | Balanced bass |
| Crest factor (average) | Input signal | 5 db | 16 db |
| | Output signal | 4 db | 11.5 db |
| Gain of BAM algorithm | | 6 db | 16 db |
| Reference | | *Figure 10* | *Figure 11* |

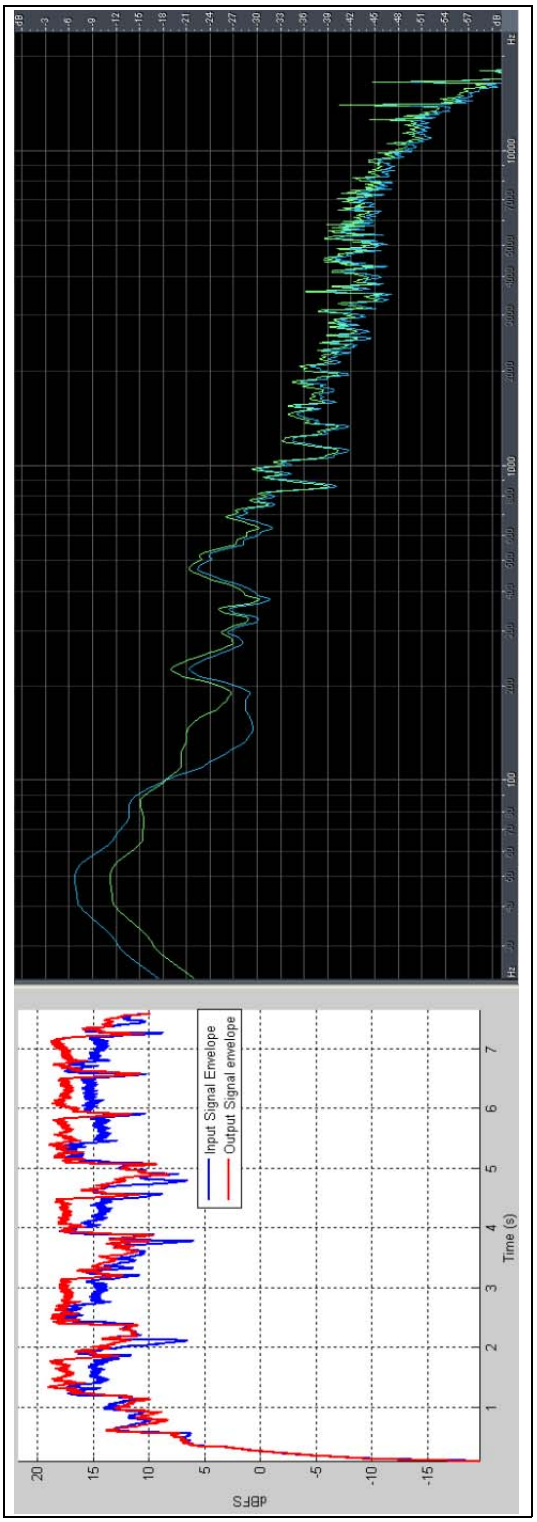



Figure 10. Hiphop music

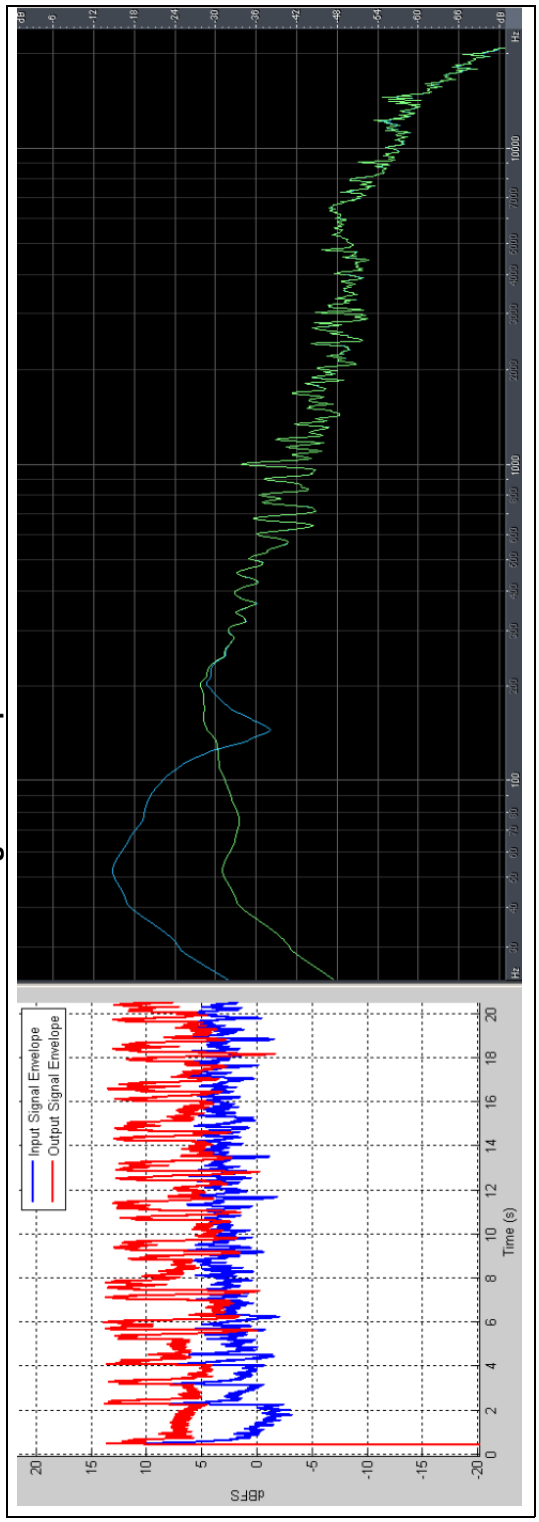



Figure 11. Pop music

# 6 Revision history

Table 13. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 26-Sep-2014 | 1 | Initial release. |
| 10-Oct-2014 | 2 | Updated document title and *Introduction*.<br>Updated title of *Table 1: Resources summary*, *Table 12: Parameters for two different music styles* and of *Figure 5: Compression curve examples*.<br>Updated *Section 1.2: Module configuration*, *Section 2.1: APIs*, *Section 2.1.3: bam_getParam function*, *Section 2.1.6: bam_process function*, *Section 2.2.1: Input and output buffers*, *Section 2.3: Static parameters structure*, *Section 3.3.1: Dynamic curves*, *Section 4.1.1: Module integration example* and *Section 4.1.2: Module APIs calls*.<br>Updated *Table 8: Input and output buffers*, *Table 9: Error values*, *Table 11: Dynamic parameters structure*, and *Table 12: Parameters for two different music styles*. |
| 27-Jan-2016 | 3 | Scope extended to STM32Cube, hence updated document title and *Introduction*.<br>Updated *Section 1.2: Module configuration*, *Section 1.3: Resources summary*, *Section 2: Module interfaces*, *Section 2.1: APIs*, *Section 2.1.3: bam_getParam function*, *Module execution*, *Section 4.1.1: Module integration example* and *Section 5: How to tune and run the application*.<br>Updated *Table 1: Resources summary*, *Table 7: Parameters for function bam_process* and *Table 9: Error values*.<br>Added *Figure 8: Audio processing chain.*<br>Updated *Figure 9: Sequence of APIs*. |
| 14-Mar-2017 | 4 | Updated *Section 1.2: Module configuration*, *Section 1.3: Resources summary*, *Section 2: Module interfaces*, *Section 2.1.1: bam_reset function*, *Section 2.1.2: bam_setParam function*, *Section 2.1.3: bam_getParam function*, *Section 2.1.4: bam_setConfig function*, *Section 2.1.5: bam_getConfig function*, *Section 2.1.6: bam_process function*, *Section 3.1: Processing steps*, *Section 4.1.1: Module integration example*, *Section 4.1.2: Module APIs calls* and *Section 5: How to tune and run the application*.<br>Updated *Table 1: Resources summary*, *Table 2: Parameters for function bam_reset*, *Table 3: Parameters for function bam_setParam*, *Table 4: Parameters for function bam_getParam*, *Table 5: Parameters for function bam_setConfig*, *Table 6: Parameters for function bam_getConfig*, *Table 7: Parameters for function bam_process*, *Table 8: Input and output buffers* and *Table 11: Dynamic parameters structure*.<br>Updated *Figure 1: BAM block diagram for mode BAM_OUTPUT_MODE_2_0*, *Figure 2: BAM block diagram for mode BAM_OUTPUT_MODE_2_1_WITH_LFE_SPLIT* and *Figure 3: BAM block diagram for mode BAM_OUTPUT_MODE_2_1_WITHOUT_LFE_SPLIT*. |
| 05-Apr-2017 | 5 | Updated *Section 1.2: Module configuration*.<br>Updated *Table 9: Error values*. |