UM2031
User manual

Sound meter library software expansion for STM32Cube

Introduction

The sound meter library software user manual describes the SoundMeterR (SMR) module configuration and its interfaces.

It describes how to integrate the module into a main program, such us the X-CUBE-AUDIO expansion software. It also provides a basic understanding of the underlying algorithm.

The sound meter library software expansion is used with 16 or 32-bit input/output format. It is part of the X-CUBE-AUDIO firmware package.

Contents

List of tables

List of figures

1 Module overview

1.1 Algorithm functionality

The SoundMeteR (SMR) module is in charge of measuring the level, on a logarithmic scale, of the incoming signal. It is based on scale conversion, smoothing filter and weighting filter.

The current implementation is using 32-bit resolution for all computations, and can be used with 16 or 32 bits input/output format. Supported sampling rates are 8 kHz, 16 kHz and 48 kHz.

1.2 Module configuration

The SMR module supports mono and stereo 16-bit or 32-bit I/O data. It is memory limited to a maximum input frame size of 960 samples, which corresponds to a 10 ms stereo signal at 48 kHz.

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

- SMR_CM4_IAR.a / SMR_CM4_GCC.a / SMR_CM4_Keil.lib: for 16 bits input/output buffers, and running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set.
- SMR_32b_CM4_IAR.a / SMR_32b_CM4_GCC.a / SMR_32b _CM4_Keil.lib: for 32 bits input/output buffers, and running on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set.
- SMR_CM7_IAR.a / SMR_CM7_GCC.a / SMR_CM7_Keil.lib: for 16 bits input/output buffers, and running on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set.
- SMR_32b_CM7_IAR.a / SMR_32b_CM7_GCC.a / SMR_32b _CM7_Keil.lib: for 32 bits input/output buffers, and running on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set.

1.3 Resource summary

Table 1 contains the module requirements for memories and frequency (MHz).

The footprints are measured on board, using IAR Embedded Workbench for ARM v7.40 (IAR Embedded Workbench common components v7.2).

Table 1. Resource summary

I/O	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM[(1)]	Frequency (MHz)
16 bits	M4	4502	8	80	392	3844	5.8
16 bits	M7	4520	8				3.9
32 bits	M4	4208	8				5.5
32 bits	M7	4624	8				3.8

1. Scratch RAM is the memory that can be shared with other process running on the same priority level.

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

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 SMR routines.

2 Module Interfaces

Two files are needed to integrate the SMR module: SMR_xxx_My.a/.lib library and the smr_glo.h header file. They 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 and must be included in the audio framework.

2.1 API

Six generic functions have a software interface to the main program:

- smr_reset function
- smr_setParam function
- smr_getParam function
- smr_setConfig function
- smr_getConfig function
- smr_process function

Each of these functions is described in the following sections.

2.1.1 smr_reset function

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

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

Table 2. smr_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 smr_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 parameters with values which cannot be changed during the module processing).

```
int32_t smr_setParam(smr_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 3. smr_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	smr_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 smr_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).

```
int32_t smr_getParam(smr_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 4. smr_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	smr_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 smr_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.

```
int32_t smr_setConfig(smr_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. smr_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	smr_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 smr_getConfig function

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

```
int32_t smr_getConfig(smr_dynamic_param_t *input_dynamic_param_ptr, void
*static_mem_ptr);
```

Table 6. smr_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	smr_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 smr_process function

This procedure is the module's main processing routine.

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

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

Table 7. smr_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. it means 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 SMR 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 followed and filled each time, before calling the processing routine, otherwise an error 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: 1 for mono, 2 for stereo
nb_bytes_per_Sample	int32_t	16-bit = 2, 32-bit = 4 SMR supports audio samples in 16-bit and 32-bit format.
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. SMR module supports only interleaved mode.

2.2.2 Returned error values

Table 9 lists the possible returned error values:

Table 9. Returned error values

Definition	Value	Description
SMR_ERROR_NONE	0	No error detected
SMR_UNSUPPORTED_INTERLEAVING_MODE	-1	If input data is not interleaved
SMR_UNSUPPORTED_NUMBER_OF_CHANNELS	-2	Input data is neither mono nor stereo
SMR_UNSUPPORTED_NUMBER_OF_BYTEPERSAMPLE	-3	Input data is neither 16-bit nor 32-bit sample format
SMR_UNSUPPORTED_AVERAGING_TIME	-4	The averaging_time is not in the following range: [0: 10000]
SMR_UNSUPPORTED_FILTER_TYPE	-5	The filter_type is not in the supported list
SMR_UNSUPPORTED_SAMPLING_RATE	-6	The sampling_rate is not equals to 8000, 16000 or 48000
SMR_BAD_HW	-7	Unsupported HW for the library

2.3 Static parameters structure

The SMR initial parameters are set using the corresponding static parameter structure before calling the smr_setParam() function.

```
struct smr_static_param {
    int32_t sampling rate;
}
typedef struct smr_static_param smr_static_param_t;
```

Table 10. Static parameters structure

Name	Type	Description
sampling_rate	int32_t	Input buffer sampling rate in Hz. Only 8 kHz, 16 kHz and 48 kHz are supported. This value is used to calculate some constant times and to initialize the filter coefficients

2.4 Dynamic parameters structure

The SMR library proposes input and output parameters in its dynamic parameters structure.

For the input parameters, it is possible to set or change the SMR configuration by setting the dynamic parameter structure before calling the smr_setConfig() function.

For the output parameters, the smr_getConfig() function must be called before accessing to the updated output dynamic parameters.

The input and output parameters are described below:

```
struct smr_dynamic_param {
    int32_t enable;              /* input variable  */
    int16_t averaging_time;      /* input variable  */
```

```
        int16_t filter_type;          /* input variable  */
        int32_t mean_level_left;      /* output variable  */
        int32_t mean_level_right;     /* output variable  */
    }
```

Table 11. Dynamic parameters structure

Name	Type	Description
enable	int32_t	1 = enable the processing of the SMR. 0 = disable the processing of the SMR, SMR output level values are not reliable.
averaging_time	int16_t	The time constant used to smooth the instantaneous level computed by the algorithm. It is expressed in ms. Minimum value is 0 ms. Maximum value is 10 000 ms.
filter_type	int16_t	Pre-filtering filter before level measurement. 4 settings are available: #define SMR_PREFILTER_NONE 0 = no pre-filter #define SMR_PREFILTER_AWEIGHTING 1 = A-weighting pre-filter #define SMR_PREFILTER_CWEIGHTING 2 = C-weighting pre-filter #define SMR_PREFILTER_DCREMOVE 3 = DC removal pre-filter
mean_level_left	int32_t	Sound level of the left (when stereo input) or mono channel. Output format is expressed in Q29.2 format.
mean_level_right	int32_t	Sound level of the right channel. When input is mono, the value is irrelevant. Output format is expressed in Q29.2 format.

3 Algorithm description

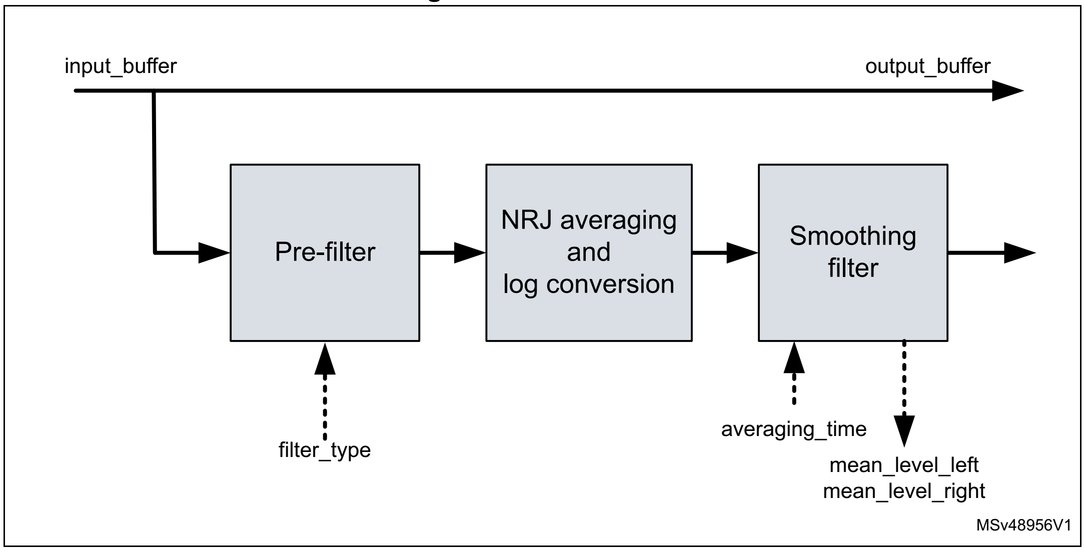
3.1 Processing steps

The SMR module proposes a measurement of the incoming signal level on a logarithmic scale. It takes as input the signal in 16 or 32 bit, mono or stereo stream and starts by applying, if configured, a pre-filter to the signal. It then computes the energy of the current frame, and converts it into a log scale. The level is then passed through a smoothing filter configured by the averaging time parameter. The output levels are then updated in the persistent memory structure, and can be accessed through the call to smr_getConfig() function and the dynamic parameters structure.

The SMR is a pass-through module, meaning the output buffer is the exact copy of the input buffer.

Figure 1 shows the SMR module.

Figure 1. SMR module



3.2 Data formats

Input of SMR module is expected to be an audio stream, mono or stereo, in 16 or 32 bit format. The algorithm can run with a maximum buffer size of 960 samples in total. For example, for a stereo signal at 48 kHz sampling rate, it will be limited to a buffer of 10ms. All operations are done with 32-bits resolution. The output format is the same as the input buffer.

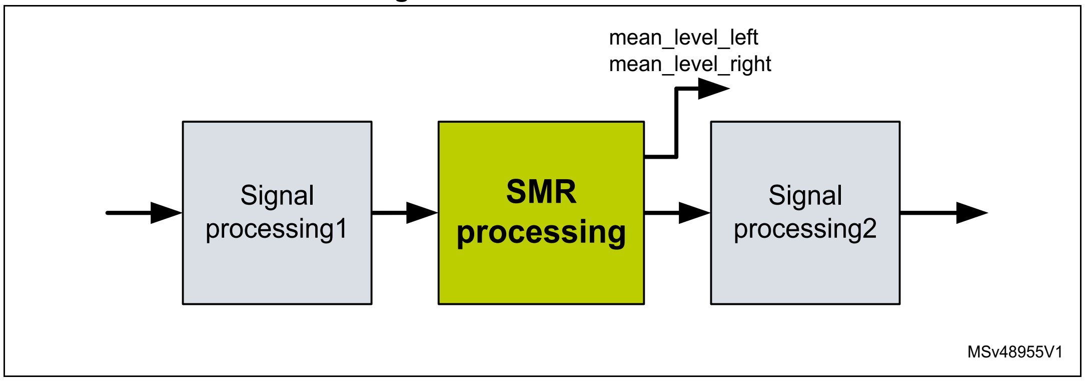
4 Application description

SMR libraries are built to run either on a Cortex® M4 or on a Cortex® M7 core. They can be integrated and run on microcontrollers of STM32F4, STM32L4, STM32F7, or STM32H7 series, respectively. There is no other hardware dependency.

4.1 Recommendations for optimal setup

The SMR module can be executed at any place in an audio processing chain as it is a pass-through module. It should be placed where the user wants to get a measurement of the signal level. Refer to *Figure 2: Basic audio chain*.

Figure 2. Basic audio chain



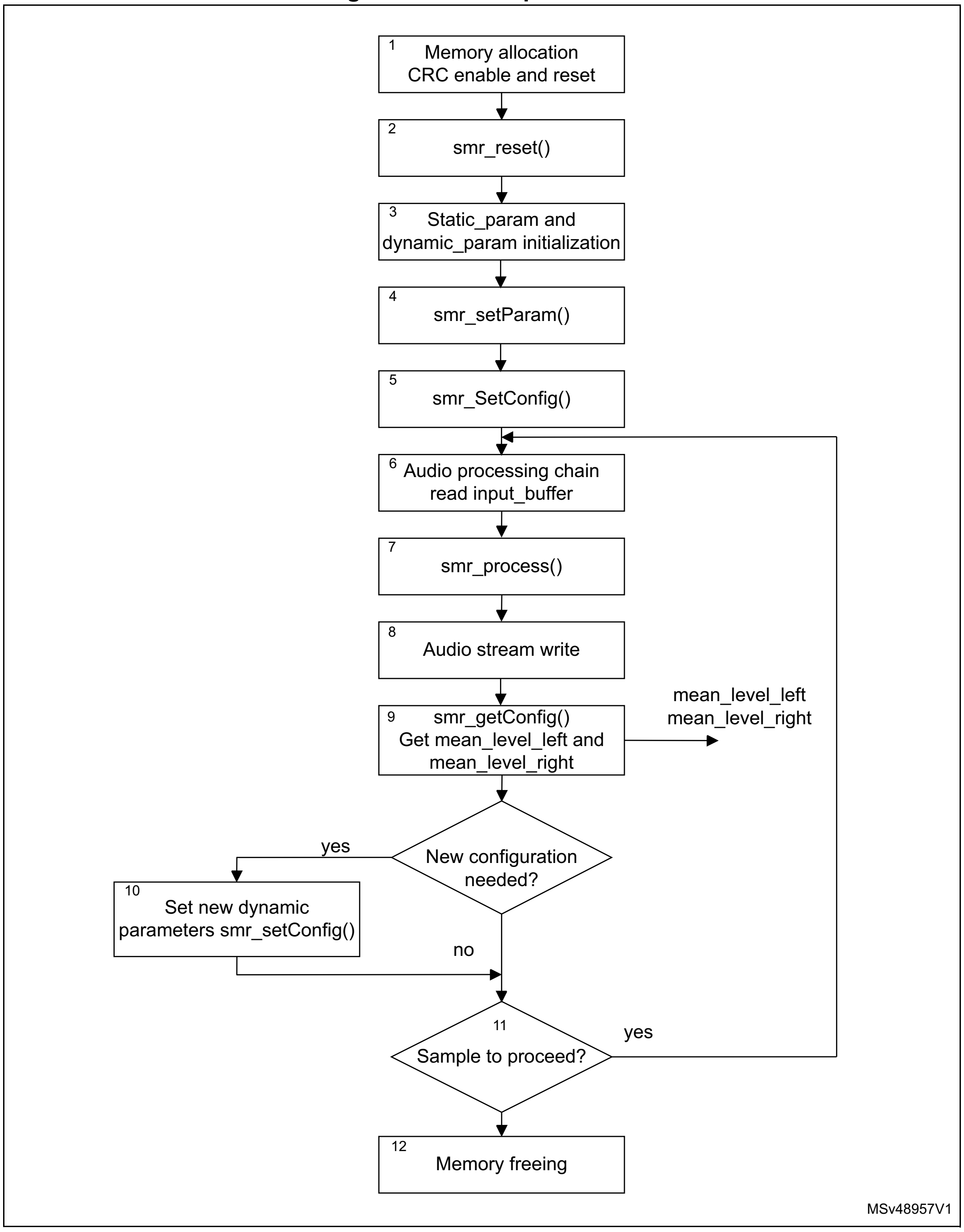
4.1.1 Module integration example

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

4.1.2 Module APIs calls

Figure 3 shows the API calls sequence.

Figure 3. API call procedure



1 Memory allocation CRC enable and reset
2 smr_reset()
3 Static_param and dynamic_param initialization
4 smr_setParam()
5 smr_SetConfig()
6 Audio processing chain read input_buffer
7 smr_process()
8 Audio stream write
9 smr_getConfig() Get mean_level_left and mean_level_right
mean_level_left mean_level_right
New configuration needed?
yes
10 Set new dynamic parameters smr_setConfig()
no
11 Sample to proceed?
yes
12 Memory freeing
MSv48957V1

1. As explained above, SMR scratch and persistent memories have to be allocated, as well as the input and output buffer.
2. Once the memory is allocated, the call to smr_reset() function will initialize the internal variables.
3. The SMR static and dynamic parameters structure can now be set.
4. Call smr_setParam() routine to apply static parameters.
5. Call to the smr_setConfig()function apply the dynamic parameters.
6. The audio stream is read from the audio 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 smr_process() function will execute the SMR algorithm.
8. The output audio stream can now be written in the proper interface. In the case of the SMR, the output buffer will be anyway the same as the input buffer.
9. In order to get the SMR mean_level_left and mean_level_right variables updated, the smr_getConfig() function has to be called before accessing the dynamic parameter structure.
10. If needed, the user can set new dynamic parameters and call the smr_setConfig() function to update module configuration.
11. 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.
12. Once the processing loop is over, the allocated memory has to be free.

5 How to tune and run the application

The SMR has few configuration parameters. However, they should be tuned with respect to what the user wants to measure.

5.1 averaging_time:

Averaging time has to be tuned accordingly to the accuracy needed by the user. Too short averaging time can lead to too much instability in the level read back, while too long averaging time can lead to hide sudden peak or drop in the incoming signal.

Sound level detectors have usually a “fast” measurement with a time averaging constant of 125 milli-second and a “slow” setting with an averaging time of 1 second.

Figure 4. Basic audio chain

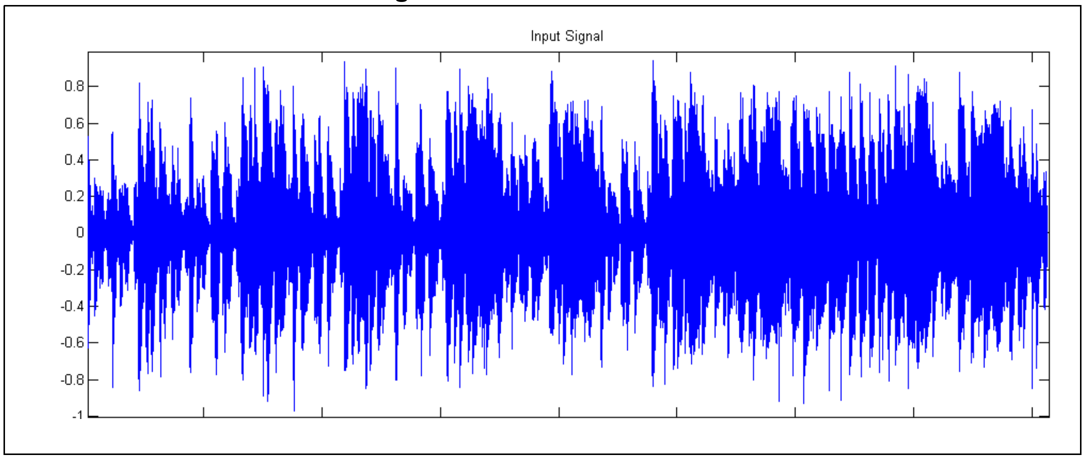
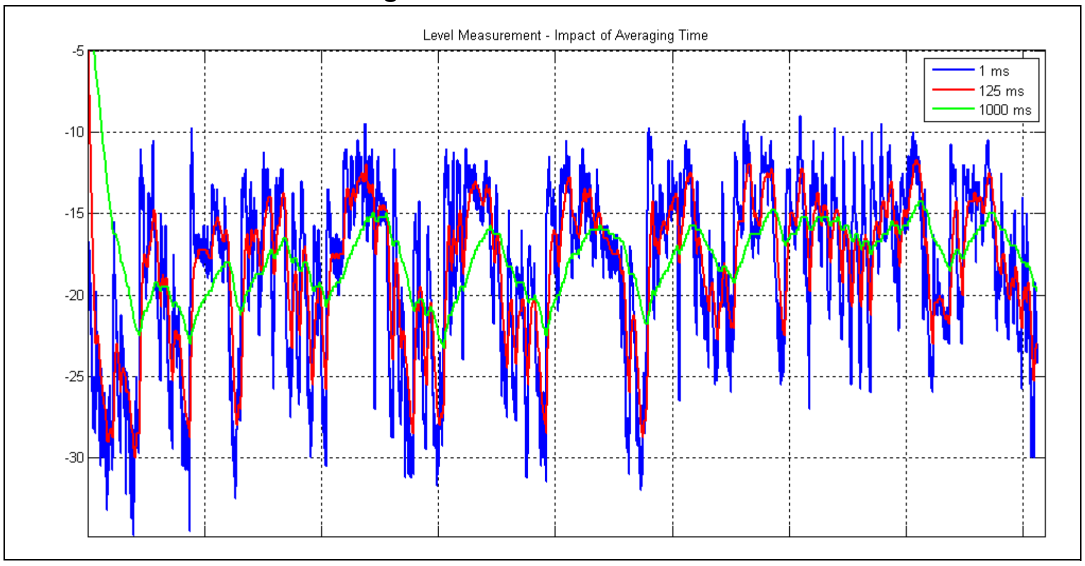


Figure 5. Level measurement



5.2 filter_type:

The DC removal filter is used when the user wants to get a measurement without any weighting on the signal spectrum, except a low-pass filter set to a very low cut-off frequency.

A-weighting filter approximates an inverted equal loudness contour at low level, while the C-weighting applies for high level. Nowadays, the A-weighting filter is the most commonly used beause it correlates well with subjective tests.

6 Revision history

Table 12. Document revision history

Date	Revision	Changes
23-Jan-2018	1	Initial release.