UM1642
User manual

Smart volume control library software expansion for STM32Cube

Introduction

The Smart Volume control (SVC) library user manual describes the software interface and requirements for integration of the module into a main program, like Audio STM32Cube expansion software and provides a rough understanding of the underlying algorithm.

The SVC 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 SVC module allows to digitally modify the volume of the input signal in the range of [-80:+36] dB.

It is based on the dynamic range compressor function, which maps the amplitude range of the audio signal to a smaller range: reducing the signal level of the higher peaks while leaving the quieter parts untreated.

1.2 Module configuration

The SVC module supports mono, stereo and multichannel interleaved 16-bit and 32-bit I/O data.

The SVC library is optimized for digital volume variations.

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

- SVC_CM4_IAR.a / SVC_CM4_GCC.a / SVC_CM4_Keil.lib: 16 bits mono/stereo and multichannel input/output buffers, library runs on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set.
- SVC_32b_CM4_IAR.a / SVC_32b_CM4_GCC.a / SVC_32b_CM4_Keil.lib: 32 bits mono/stereo and multichannel input/output buffers, library runs on any STM32 microcontroller featuring a core with Cortex®-M4 instruction set.
- SVC_CM7_IAR.a / SVC_CM7_GCC.a / SVC_CM7_Keil.lib: 16 bits mono/stereo and multichannel input/output buffers, library runs on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set.
- SVC_32b_CM7_IAR.a / SVC_32b_CM7_GCC.a / SVC_32b_CM7_Keil.lib: 32 bits mono/stereo and multichannel input/output buffers, library runs on any STM32 microcontroller featuring a core with Cortex®-M7 instruction set.

1.3 Resource summary

Table 1 contains the module requirements for the Flash, stack and RAM memories and the frequency (MHz).

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

Table 1. Resource summary

Version	Use Case	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM	Frequency (MHz)
16 bits I/O stereo	High Quality	M4	6276 Bytes	8 Bytes	74 Bytes	1368 Bytes	2880 Bytes	10.7
		M7	**8220 Bytes**					**7.9**
	standard	M4	6276 Bytes					5.9
		M7	**8220 Bytes**					**4**
16 bits I/O multichannel 3.1	High Quality	M4	6276 Bytes	8 Bytes	74 Bytes	1368 Bytes	2880 Bytes	15.6
		M7	**8220 Bytes**					**10.6**
	standard	M4	6276 Bytes					9.9
		M7	**7752 Bytes**					**6.1**
32 bits I/O stereo	High Quality	M4	6152 Bytes	8 Bytes	74 Bytes	2648 Btyes	4800 Bytes	10.9
		M7	**7752 Bytes**					**7.2**
	standard	M4	6152 Bytes					6.2
		M7	**7752 Bytes**					**3.4**
32 bits I/O multichannel 3.1	High Quality	M4	6152 Bytes	8 Bytes	74 Bytes	2648 Btyes	4800 Bytes	14.2
		M7	**7752 Bytes**					**9.4**
	standard	M4	6152 Bytes					8.3
		M7	**7752 Bytes**					**4.6**

Note: The 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 SVC routines.

2 Module interfaces

Two files are needed to integrate the SVC module. SVC_xxx_CMy_zzz.a/.lib library and *the svc_glo.h* header file which contains 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:

- svc_reset
- svc_setParam
- svc_getParam
- svc_setConfig
- svc_getConfig
- svc_process

2.1.1 svc_reset function

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

API description:

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

Table 2. svc_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

Note: *This routine must be called at least once at initialization time, when the real time processing has not started.*

2.1.2 svc_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 (frame after frame).

Note: *Static parameters cannot be changed dynamically after the start of the module processing, while dynamic parameters can be modified during processing (through svc_setConfig() API described below).*

API description:

```
int32_t svc_setParam(svc_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 3. svc_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	svc_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 svc_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, that is the parameters with values which cannot be changed during the module processing (frame after frame).

API description:

```
int32_t svc_getParam(svc_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 4. svc_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	svc_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 svc_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 (after reset and setParam routines).

API description:

```
int32_t svc_setConfig(svc_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. svc_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	svc_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 svc_getConfig function

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

API description:

```
int32_t svc_getConfig(svc_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 6. svc_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	svc_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 svc_process function

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

API description:

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

Table 7. svc_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

Note: This process routine can run in place; the input_buffer and the output_buffer addresses can be the same.

2.2 External definitions and types

For genericity reasons and to facilitate the integration in main frameworks, some types and definitions are adopted.

2.2.1 Input and output buffers

The SVC 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 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: 1 for mono, 2 for stereo, 4 for 3.1 multichannel,...
nb_bytes_per_Sample	int32_t	Dynamic of data in number of bytes (16-bit = 2, 32-bit = 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 describes possible returned error values.

Table 9. Returned error values

Definition	Value	Description
SVC_ERROR_NONE	0	Ok - No error detected
SVC_UNSUPPORTED_DELAY_LENGTH	-1	Delay length is not supported
SVC_UNSUPPORTED_VOLUME	-2	Volume setting is outside [-80;36] dB range
SVC_UNSUPPORTED_MUTE_MODE	-3	Mute value is not supported
SVC_UNSUPPORTED_QUALITY_MODE	-4	High quality value is not supported
SVC_UNSUPPORTED_JOINT_STEREO_MODE	-5	Joint stereo value is not supported
SVC_UNSUPPORTED_NUMBER_OF_BYTEPERSAMPLE	-6	Input data is neither 16-bits nor 32-bits format
SVC_BAD_HW	-7	Unsupported hardware for the library

2.3 Static parameters structure

There are some static parameter to be set before calling the process routine.

```
struct svc_static_param {
   int16_t delay_len;
   int16_t joint_stereo;
};
typedef struct svc_static_param svc_static_param_t;
```

Table 10. Static parameters structure

Name	Type	Description
delay_len	int16_t	Delay introduced in number of samples in the range [0:160]
joint_stereo	int16_t	1 to enable joint_stereo, 0 to disable it

- "Delay_len" parameter represents the delay applied between the gain computed from an input sample and this input sample.

  For tuning, it is advised to smoothen the signal peaks and to use the following value:

  delay_len = attack_time * Fs * 2/3
- "Joint_stereo" parameter allows to apply the same amount of gain reduction to all the channels: left, right, and possibly other channels in case of multichannel, in order to preserve the stereo image of the input signal.

  As it almost divides the MHz consumption of the algorithm by 2, it is highly recommended to set it.

The default settings are:
- Delay len is set to 100 samples
- Joint_stereo is enabled.

2.4 Dynamic parameters structure

There are two dynamic parameters to be used.

```
struct svc_dynamic_param {
   int16_t   target_volume_dB;
   int16_t   mute;
   int16_t   enable_compr;
   int32_t   attack_time;
   int32_t   release_time;
   int16_t   quality;
};
typedef struct svc_dynamic_param svc_dynamic_param_t;
```

Table 11. Dynamic parameters structure

Name	Type	Description
target_volume_dB	int16_t	Volume dB input value, in 1/2 dB steps
mute	int16_t	1 to enable mute, 0 to disable it
enable_compr	int16_t	1 to enable compression, 0 to disable it. It is highly recommended to enable the compression to avoid saturation with positive gains.
attack_time	int32_t	Attack time coefficient in Q31 format
release_time	int32_t	Release time coefficient in Q31 format
quality	int16_t	1 to enable HIGH_Q mode, 0 to enable standard mode

High quality

This configuration computes the gain to apply for each input sample, and gives the best possible results of the module. This has to be compared with the "Standard" version which computes the gain to apply for a block of 16 input samples, considering their mean value. The "Standard" configuration is less reactive and can increase the clipping samples percentage as compared to the "High quality" configuration; but it consumes twice less MHz.

Attack and release times

Conversion formula: Attack_time α_A and release time α_R are the coefficients given as input of the algorithm, in Q31 format, corresponding to the following formula:

$$\alpha = (e^{(-1)/(\tau \cdot Fs)} \cdot 2^{31})$$

Where:

Fs = 48 kHz and τ is the attack/release time in seconds.

Value Range: τ_{MIN} = 1 sample @Fs = 48 kHz = 0.02 ms (physical limit)

τ_{MAX}: there is no maximum limitation, whereas values exceeding tens of second do not seem realistic. The implementation limit is 2^{31} because the parameter is a 32-bit format.

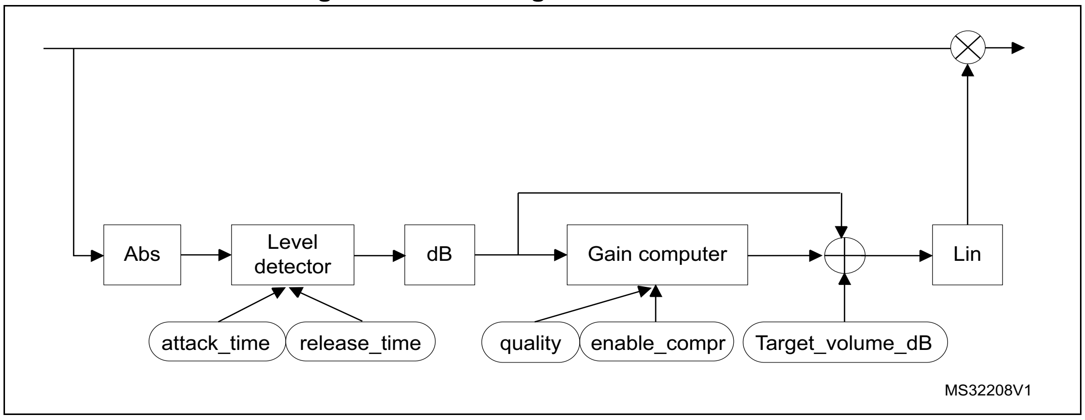
The conversion between τ and α must be done outside the algorithm, by the framework. α_A and α_R must be given as "attack_time" and "release_time" inputs to the algorithm.

Note: Considering τ range described above, α_A and α_R should be in the following range: [790015084:2147483647]. Anyway, a tuning interface would propose τ to be set between 0.02ms and 5ms for the attack, and between 50ms and 300ms for the release times.

3 Algorithm description

3.1 Processing steps

Figure 1. Block diagram of SVC module



Abs block: computes the absolute value of the input signal.

Level detector block: smoothens the signal depending on attack and release times.

dB block: converts the signal to dB format.

Gain computer block: computes in real time the optimized compression curve depending on the targeted volume.

Lin block: converts the signal back to the linear format.

3.2 Data formats

The SVC module supports fixed point mono, stereo and multichannel interleaved 16-bit and 32-bit input data.

It can work independently of the frame size and the signal input sampling rate.

However it is recommended to avoid selecting very short frame size (ie 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 the 16-bits or 32-bits resolution, as the input data.

3.3 Performance assessment

3.3.1 Compression gain

The compression gain curve design, which is calculated from the input volume, is an essential part of the SVC module. Therefore, its real time computation must be very accurate.

Find below the compression gain curve obtained with the Matlab model and with the SVC module for 6 dB (see *Figure 2*) and 20 dB (see *Figure 3*) input volumes. The x-axis represents the dB volume of the input sample, and the y-axis represents the dB volume applied to the output sample and computed by the algorithm. You can notice two parts in the red curve: a linear part for low volume inputs, and a compressed part for higher volume inputs which is needed to prevent the output from clipping.

Note: The blue curve noted "Input" represents the compression gain = 1, i.e. when output = input.

Figure 2. SVC compression curve with 6 dB input volume

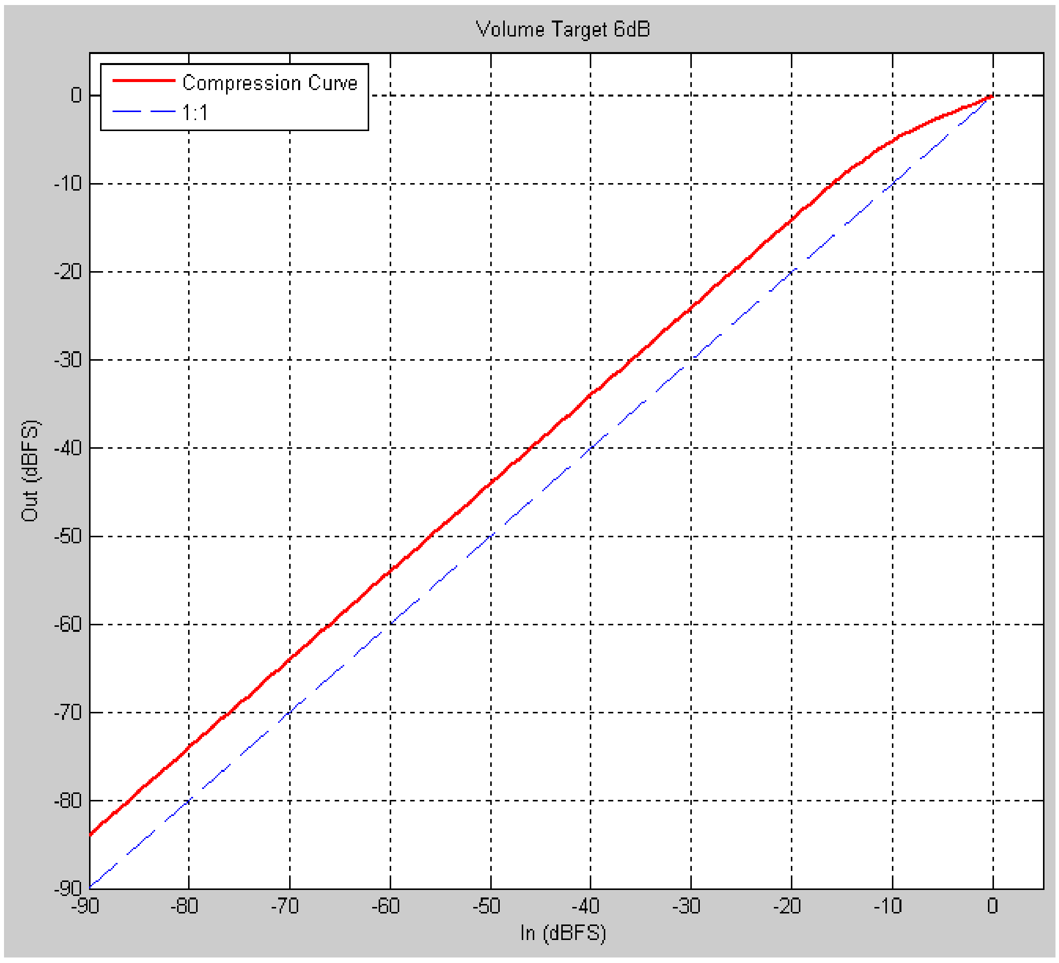
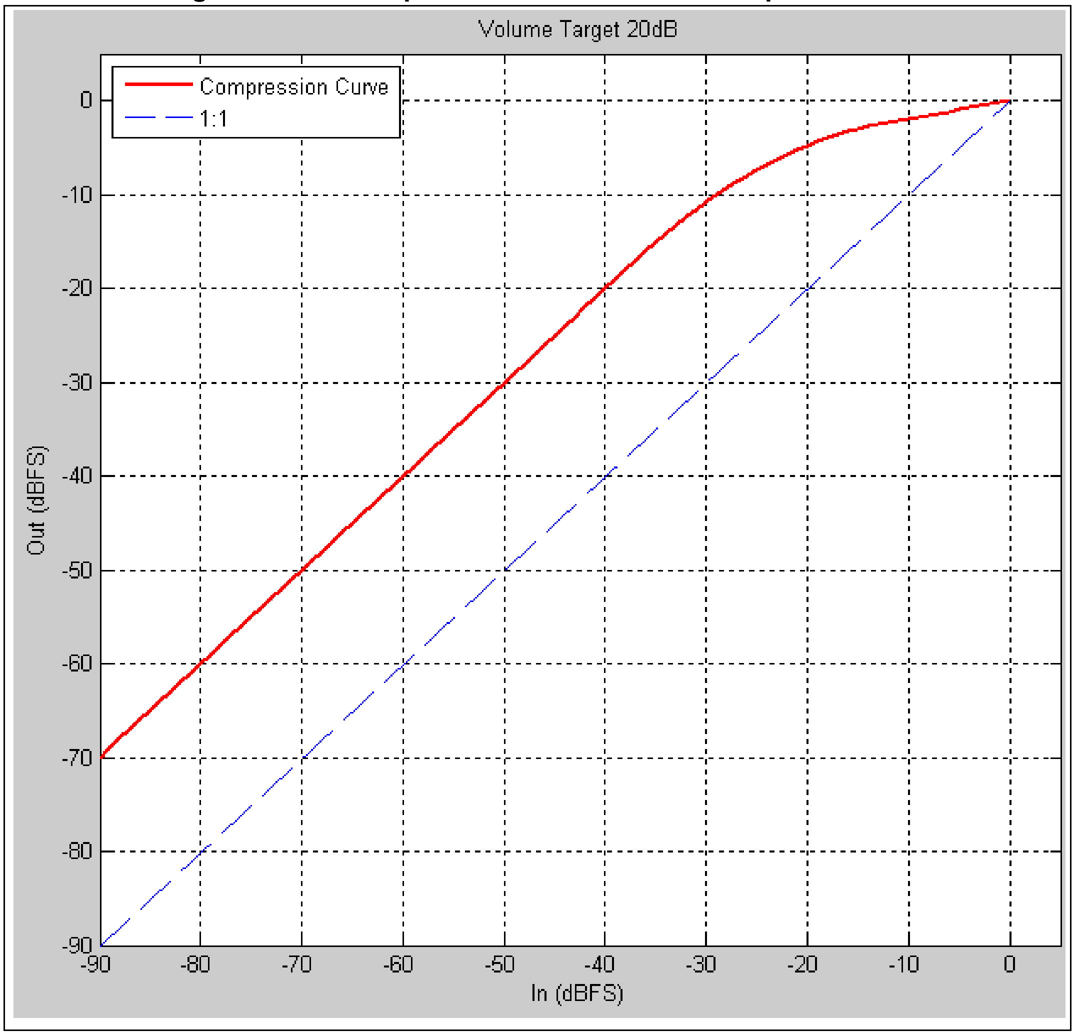


Figure 3. SVC compression curve with 20 dB input volume



3.3.2 Total Harmonic Distortion (THD)

THD, Total Harmonic Distortion, is a measurement used to characterize the linearity of audio systems.

When the input is a pure sine wave, the measurement is most commonly the ratio of the sum of the powers of all harmonic frequencies to the power of the fundamental frequency:

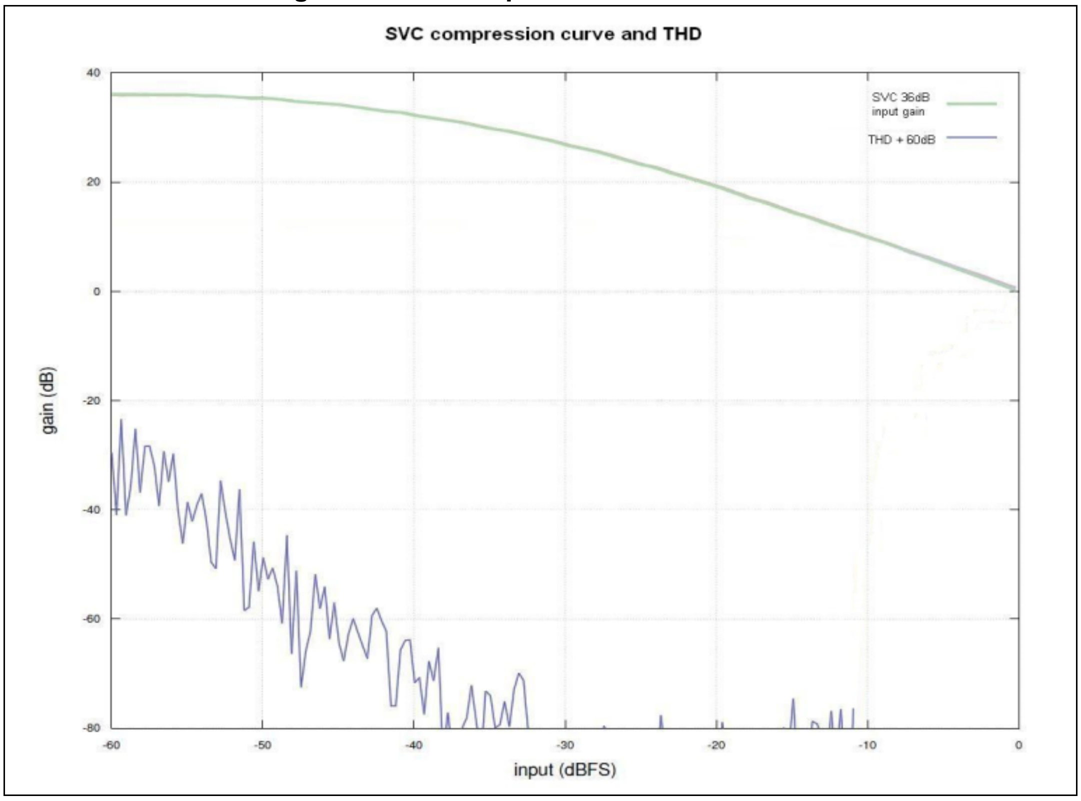
$$\mathrm{THD} = \frac{P2 + P3 + P4 + \ldots + P\infty}{P1} = \frac{\sum_{i=2}^{\infty} Pi}{P1}$$

As compressors are non-linear modules, it is useful to know which distortion the SVC module can add to the signal.

In *Figure 4*, the blue curve represents the THD value measured for SVC with 36 dB input gain. The THD increases as expected in the non-linearity part of the compression curve, and especially when input > -12 dB. The green curve represents the compression gain of the SVC with 36 dB input gain.

Note: The value represented is THD +60 dB, thus a gain of 0 dB on the graph corresponds to -60 dB.

Figure 4. SVC compression curve and THD



3.3.3 Amplitude statistics

Amplitude statistics are important measures to evaluate the audio signal characteristics.

In order to assess the performance of the SVC implementation, several measures have been identified: the number of clipped samples, the RMS power and the stereo image of each channel.

Table 12 is an example of comparison between "Joint stereo" and "No Joint stereo" modes. It also shows the differences between the "high quality" and "standard" configurations for the same input gain = 36 dB.

The results shown in *Table 12* are obtained with the same input file: "miossec" for several input gains: 6, 12, 18, 24, and 36 dB.

You can observe that the RMS of the "Joint stereo" mode is lower than the "No joint stereo" one. The difference depends on the input gain (from 0.1 to 1 dB). Consequently, the clipping percentage is also lower for the "Joint stereo" mode.

Knowing that the input signal has a stereo image of 0.54 dB, the "Joint stereo" allows to preserve the stereo image for all input gains, whereas "No joint stereo" does not.

Table 12. Amplitude statistics

Input file	No joint stereo			Joint Stereo			DIFF Clipped samples %	DIFF RMS	Stereo Image No Joint	Stereo Image Joint
	Clipped samples	Clipped samples %	Total RMS Power (dB)	Clipped samples	Clipped samples %	Total RMS Power (dB)				
Miossec_6dB_L	137	0.00809	-11.01	93	0.00549	-11.21	0.00256	0.2	–	–
Miossec_6dB_R	125	0.00738	-10.56	97	0.00573	-10.67	0.00166	0.11	0.45	0.54
Miossec_12dB_L	922	0.05446	-7.88	560	0.03308	-8.46	0.02138	0.58	–	–
Miossec_12dB_R	986	0.05824	-7.6	693	0.04094	-7.92	0.01731	0.32	0.28	0.54
Miossec_18dB_L	4615	0.27259	-6.12	2251	0.13296	-6.94	0.13963	0.82	–	–
Miossec_18dB_R	5222	0.30844	-5.96	3351	0.19793	-6.4	0.11051	0.44	0.16	0.54
Miossec_24dB_L	13535	0.79945	-5.11	5909	0.34902	-6.07	0.45043	0.96	–	–
Miossec_24dB_R	14641	0.86478	-5.02	9317	0.55031	-5.53	0.31446	0.51	0.09	0.54
Miossec_30dB_L	25898	1.52968	-4.51	11066	0.65362	-5.53	0.87606	1.02	–	–
Miossec_30dB_R	27374	1.61686	-4.46	17561	1.03725	-5.09	0.57961	0.63	0.05	0.44
Miossec_36dB_L	40418	2.38731	-4.08	17769	1.0495	-5.12	1.33777	1.04	–	–
Miossec_36dB_R	41836	2.47107	-4.05	27608	1.6307	-4.6	0.84038	0.55	0.03	0.52
Miossec_36dB STANDARD_L	109420	6.46295	-3.1	87411	5.16298	-3.5	1.29997	0.4	–	–
Miossec_36dB STANDARD_R	102256	6.03980	-3.21	111142	6.56466	-3.07	-0.52485	-0.14	-0.11	0.43

4 System requirements and hardware setup

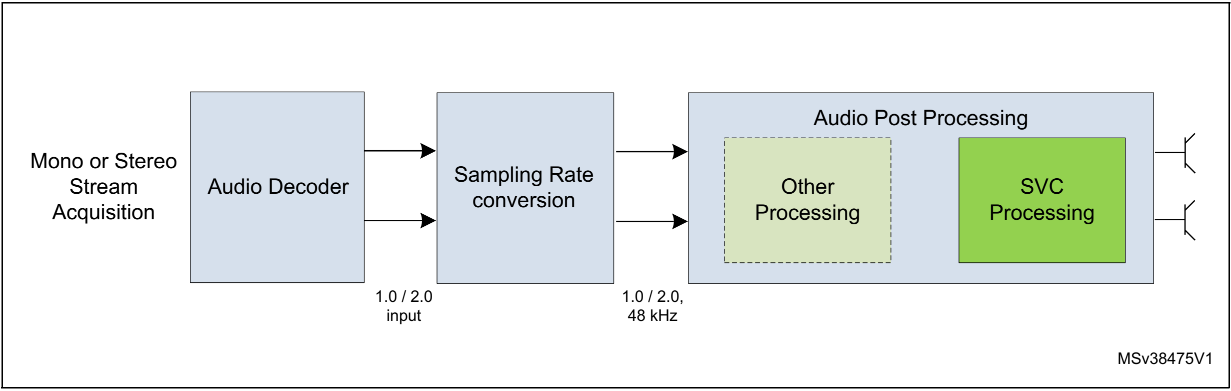
SVC libraries are built to run either on a Cortex® M4 or on a Cortex® M7 core, without FPU usage. They can be integrated and run on corresponding STM32F4/STM32L4 or STM32F7 family devices.

4.1 Recommendations for an optimal setup

The SVC library should be executed close to the audio DAC, at the end of the chain, because of its non-linear effect on the signal.

Refer to *Figure 5: Basic Audio Chain*

Figure 5. Basic Audio Chain

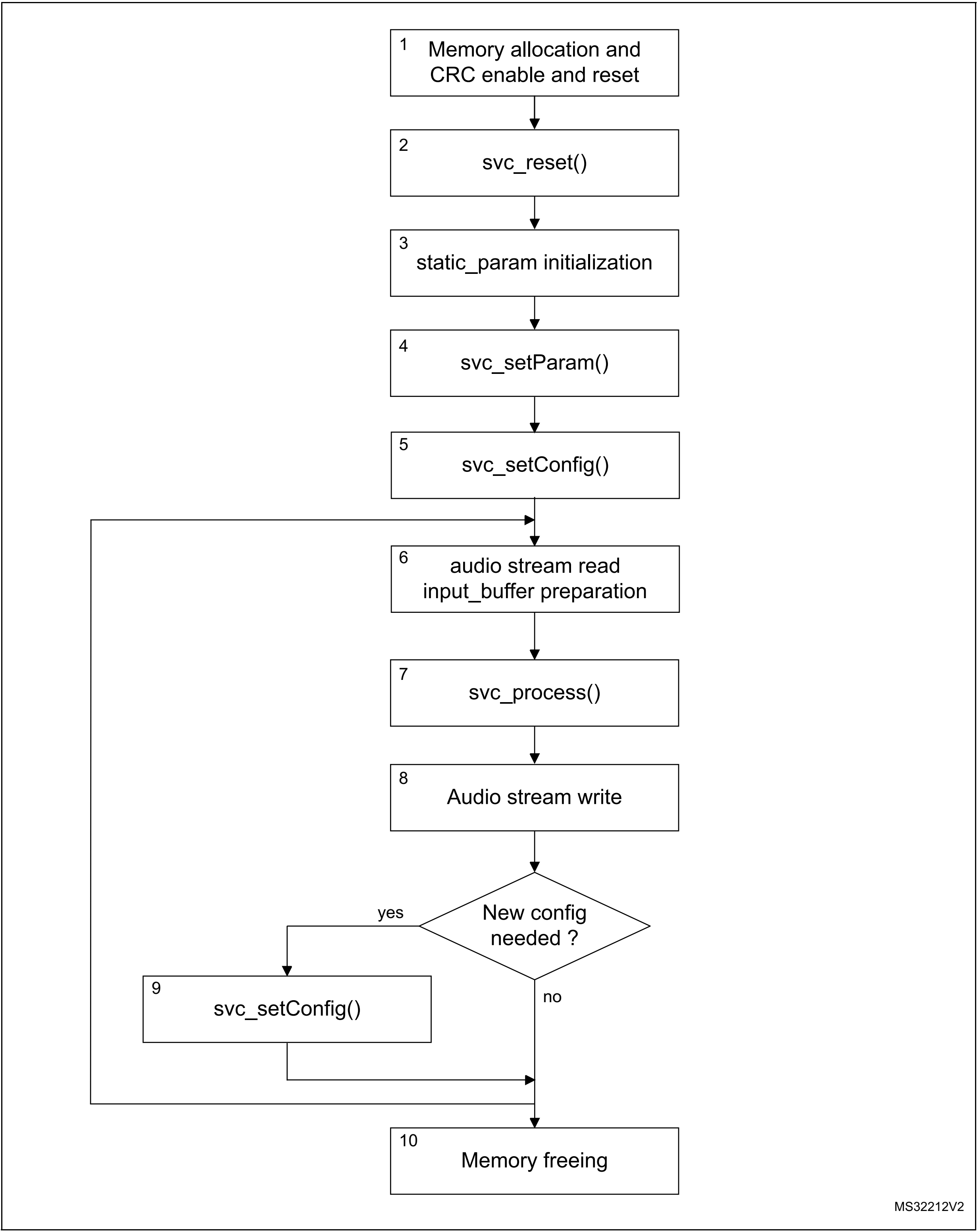


4.1.1 Module integration example

Cube expansion SVC integration examples are provided on STM32F746G-Discovery and STM32F469I_Discovery boards. Please refer to provided integration code for more details.

4.1.2 Module integration summary

Figure 6. API call procedure



1. As explained above, the SVC scratch and persistent memories have to be allocated, as well as the input and output buffers. Also, SVC library must run on STM32 devices so CRC HW block must be enable and reset.
2. Once the memory is allocated, the call to the svc_reset() function initializes the internal variables and sets the default configuration to the module.
3. The module static configuration can now be set by initializing the static_param structure.
4. Call the svc_setParam() routine to send static parameters from the audio framework to the module.
5. Get dynamic parameters when they are updated, and call svc_setConfig() routine to send dynamic parameters from the audio framework to the module.
6. 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.
7. Call the main processing routine to apply the effect.
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 svc_setConfig() routine again, to update the module configuration.
10. Once the processing loop is over, the allocated memory has to be freed.

5 How to tune and run the application

Once the module is integrated into the audio framework to play samples at 48 kHz, just launch the Audio player and choose a .WAV or .MP3 file with a 48 kHz sampling frequency if there is no sampling rate conversion available.

The SVC "target_volume_dB" dynamic parameter represents the volume asked by the user, in half dB steps. The volume change is applied smoothly to avoid audible artifacts.

The SVC "attack_time" dynamic parameter represents the time in ms, to decrease the gain, when the input overshoots the threshold. The lowest the attack time is, the quickest the gain will follow the input signal when the input signal amplitude increases.

The SVC "release_time"dynamic parameter represents the time in ms, to bring back the gain to a normal value, once the signal falls below the threshold. The lowest the release time is, the quickest the gain will follow the input signal when the input signal amplitude decreases.

The SVC "enable_compr" dynamic parameter enables the effect when set to 1 or disables it when set to 0.

The SVC "mute" dynamic parameter mutes the output when set to 1 or has no influence on input signal when set to 0. When enabled, it allows mute the signal smoothly over a frame, avoiding audible artifacts.

The SVC "quality" dynamic parameter, when enabled, allows to compute the gain for each sample. On the contrary, when disabled, the gain is computed with an average over 16 input samples, to reduce MHz consumption.

6 Revision history

Table 13. Document revision history

Date	Revision	Changes
18-Jul-2013	1	Initial release.
08-Nov-2013	2	Added 32-bit I/O data and multichannel support. Updated *Section 1.2: Module configuration*, *Section 2.3: Static parameters structure*, *Section 3.2: Data formats* and *Section 3.3.1: Compression gain*. Updated *Table 2: Resource summary*. Updated *Figure 2: SVC compression curve with 6 dB input volume* and *Figure 3: SVC compression curve with 20 dB input volume*.
28-Nov-2014	3	Updated RPN on cover page
10-Dec-2014	4	Updated *Section 4.2.2* and *Section 5*
24-Feb-2016	5	Updated: – *Introduction*, *Section 1.2*, *Section 2.1*, *Section 3.2*, *Section 4.1*, *Section 5* – *Table 1*, *Table 9*, *Table 10*, *Table 12* – *Figure 4*, *Figure 6* Added: – *Figure 5*
21-Mar-2017	6	Updated: – *Table 1: Resource summary*, *Table 2: svc_reset*, *Table 3: svc_setParam*, *Table 4: svc_getParam*, *Table 5: svc_setConfig*, *Table 6: svc_getConfig*, *Table 7: svc_process*, *Table 8: Input and output buffers*, *Table 11: Dynamic parameters structure*, – *Section 1.2: Module configuration*, *Section 1.3: Resource summary*, *Section 2: Module interfaces*, *Section 2.1.1: svc_reset function*, *Section 2.1.3: svc_getParam function*, *Section 2.1.4: svc_setConfig function*, *Section 2.1.5: svc_getConfig function*, *Section 2.1.6: svc_process function*, *Section 2.3: Static parameters structure*, *Section 2.4: Dynamic parameters structure*, *Section 4.1.1: Module integration example*, *Section 4.1.2: Module integration summary*, *Section 5: How to tune and run the application*
10-Jan-2018	7	Replaced X-CUBE-AUDIO-F4, X-CUBE-AUDIO-F7 and X-CUBE-AUDIO-L4 with X-CUBE-AUDIO.