UM2024
User manual

Headphone virtualization library software expansion for STM32Cube

Introduction

The HeadPhone Virtualization (HPV) library user manual describes the software interface and requirements for the integration of the module into a main program like the Audio STM32Cube expansion software and provides a rough understanding of the underlying algorithm.

The HPV library implements the audio virtualization for headphone from mono to 7.1 input signals. Please refer to UM1655 and UM1633 for loudspeakers virtualization.

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

Contents

List of tables

List of figures

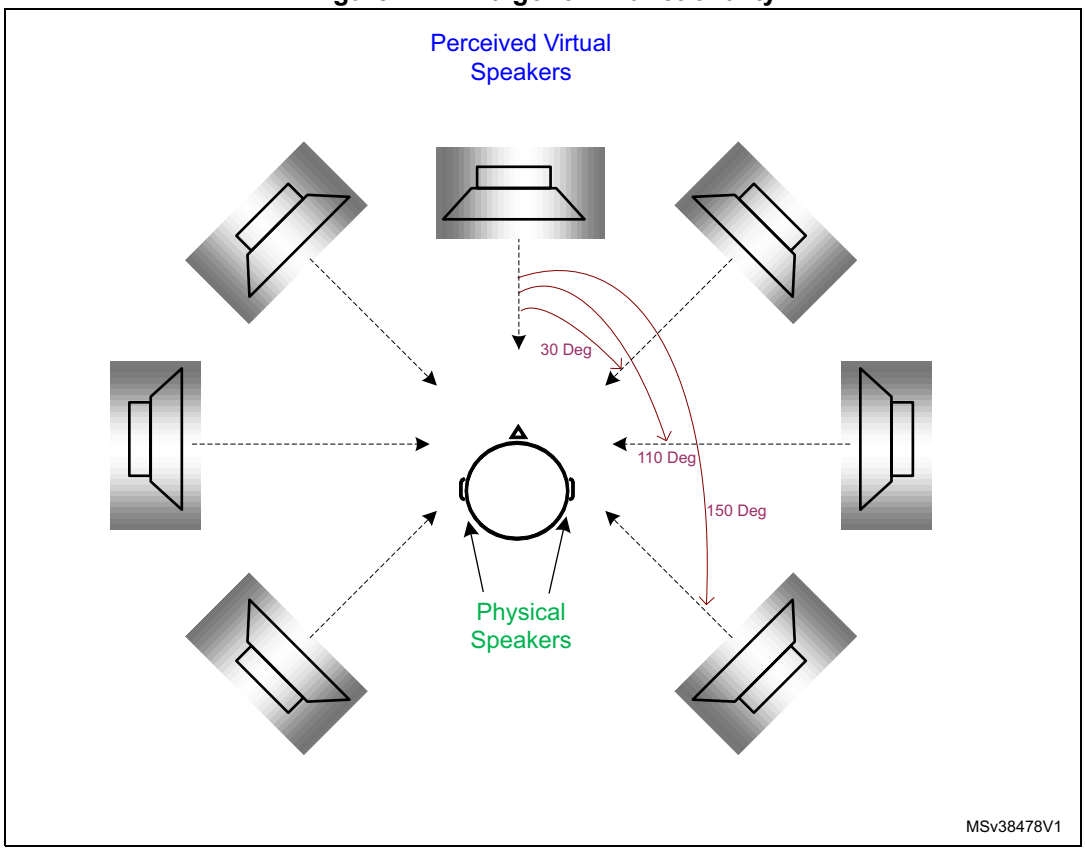
1 Module overview

1.1 Algorithm function

The HPV module provides functions to handle audio virtualization on headphones. Goals of this module are to have virtualization effect and to feel sounds less aggressive for the ears. Front channels (from stereo or multichannel files) are no more heard on the sides close to the ears but are narrowed in front of the listener, while surround and rear sounds are played around the listener. Some reverberation and “Out of Head” effects help feeling greater virtualization and allows to feel smoother sounds as well (attenuation of HP vibrations).

The *Figure 1* presents the effect perception with HPV activated.

Figure 1. HPV algorithm functionality



1.2 Module configuration

The HPV module supports 1.0, 2.0, 5.1 and 7.1 interleaved 16-bit and 32-bit I/O data at a 48 kHz sampling frequency, with a minimum input frame size of 2ms (96 samples per channel). For MIPS optimization reasons, frame length must be a multiple of 4 samples as well.

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

- HPV_20_CM4_IAR.a / HPV_20_CM4_GCC.a / HPV_20_CM4_Keil.lib: standard version optimized for mono and stereo inputs only, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_20_HQ_CM4_IAR.a / HPV_20_HQ_CM4_GCC.a / HPV_20_HQ_CM4_Keil.lib: High Quality version optimized for mono and stereo inputs only, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_20_HQ_32b_CM4_IAR.a / HPV_20_HQ_32b_CM4_GCC.a / HPV_20_HQ_32b_CM4_Keil.lib: High Quality version optimized for mono and stereo inputs only, with 32 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_CM4_IAR.a / HPV_CM4_GCC.a / HPV_CM4_Keil.lib: standard version optimized for mono, stereo, 5.1 and 7.1 inputs, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_HQ_CM4_IAR.a / HPV_HQ_CM4_GCC.a / HPV_HQ_CM4_Keil.lib: High Quality version optimized for mono, stereo, 5.1 and 7.1 inputs, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_HQ_32b_CM4_IAR.a / HPV_HQ_32b_CM4_GCC.a / HPV_HQ_32b_CM4_Keil.lib: High Quality version optimized for mono, stereo, 5.1 and 7.1 inputs, with 32 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M4 instruction set.
- HPV_20_CM7_IAR.a / HPV_20_CM7_GCC.a / HPV_20_CM7_Keil.lib: standard version optimized for mono and stereo inputs only, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.
- HPV_20_HQ_CM7_IAR.a / HPV_20_HQ_CM7_GCC.a / HPV_20_HQ_CM7_Keil.lib: High Quality version optimized for mono and stereo inputs only, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.
- HPV_20_HQ_32b_CM7_IAR.a / HPV_20_HQ_32b_CM7_GCC.a / HPV_20_HQ_32b_CM7_Keil.lib: High Quality version optimized for mono and stereo inputs only, with 32 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.
- HPV_CM7_IAR.a / HPV_CM7_GCC.a / HPV_CM7_Keil.lib: standard version optimized for mono, stereo, 5.1 and 7.1 inputs, with 16 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.
- HPV_HQ_CM7_IAR.a / HPV_HQ_CM7_GCC.a / HPV_HQ_CM7_Keil.lib: High Quality version optimized for mono, stereo, 5.1 and 7.1 inputs, with 16 bits input/output buffers and it run on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.
- HPV_HQ_32b_CM7_IAR.a / HPV_HQ_32b_CM7_GCC.a / HPV_HQ_32b_CM7_Keil.lib: High Quality version optimized for mono, stereo, 5.1 and 7.1 inputs, with 32 bits input/output buffers and it runs on any STM32 microcontroller featuring a core with Cortex-M7 instruction set.

1.3 Resources summary

Table 1 contains Flash, stack, RAM and frequency requirements, the consumed MHz being measured with a 10 ms framing.

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

-	Use case @ 48 kHz	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM	Frequency (MHz)
HPV Stereo Standard Quality, 16-bits I/O	1.0=>2.0	M4	11142 Bytes	7800 Bytes	500 Bytes	17220 Bytes	2884 Bytes	27
		M7	**11314 Bytes**					**18.9**
	2.0=>2.0	M4	11142 Bytes					25
		M7	**11314 Bytes**					**16.3**
HPV Stereo High Quality, 16-bits I/O	1.0=>2.0	M4	10166 Bytes	7800 Bytes	500 Bytes	19908 Bytes	3460 Bytes	26
		M7	**10178 Bytes**					**17.7**
	2.0=>2.0	M4	10166 Bytes					23
		M7	**10178 Bytes**					**14.9**
HPV Multi Standard Quality, 16-bits I/O	1.0=>2.0	M4	11674 Bytes	7800 Bytes	500 Bytes	26180 Bytes	3844 Bytes	27
		M7	**11798 Bytes**					**19**
	2.0=>2.0	M4	11674 Bytes					24
		M7	**11798 Bytes**					**16.3**
	5.1=>2.0	M4	11674 Bytes					76
		M7	**11798 Bytes**					**48.3**
	7.1=>2.0	M4	11674 Bytes					86
		M7	**11798 Bytes**					**51.4**

Table 1. Resources summary (continued)

-	Use case @ 48 kHz	Core	Flash code (.text)	Flash data (.rodata)	Stack	Persistent RAM	Scratch RAM	Frequency (MHz)
HPV Multi High Quality, 16-bits I/O	1.0=>2.0	M4	11098 Bytes	7800 Bytes	500 Bytes	35588 Bytes	5380 Bytes	27
		M7	**11126 Bytes**					**19.8**
	2.0=>2.0	M4	11098 Bytes					24
		M7	**11126 Bytes**					**16.9**
	5.1=>2.0	M4	11098 Bytes					75
		M7	**11126 Bytes**					**52.5**
	7.1=>2.0	M4	11098 Bytes					86
		M7	**11126 Bytes**					**56.6**
HPV Stereo High Quality, 32-bits I/O	1.0=>2.0	M4	10166 Bytes	7800 Bytes	500 Bytes	19908 Bytes	3460 Bytes	28
		M7	**10178 Bytes**					**18**
	2.0=>2.0	M4	10166 Bytes					24.5
		M7	**10178 Bytes**					**14.9**
HPV Multi High Quality, 32-bits I/O	1.0=>2.0	M4	11098 Bytes	7800 Bytes	500 Bytes	35508 Bytes	5380 Bytes	28
		M7	**11126 Bytes**					**19.6**
	2.0=>2.0	M4	11098 Bytes					24.5
		M7	**11126 Bytes**					**16.8**
	5.1=>2.0	M4	11098 Bytes					79
		M7	**11126 Bytes**					**52.3**
	7.1=>2.0	M4	11098 Bytes					93
		M7	**11126 Bytes**					**56.4**

Note: Footprints on STM32F7 are measured on boards with stack, persistent and scratch RAM located in DTCM memory while I/O buffers are located in ISRAM.
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 HPV routines.

2 Module Interfaces

Two files are needed to integrate HPV module. *HPV_xxx_CMy_zzz.a/.lib* library and the *hpv_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:

- hpv_reset
- hpv_setParam
- hpv_getParam
- hpv_setConfig
- hpv_getConfig
- hpv_process

2.1.1 hpv_reset function

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

```
int32_t hpv_reset(void *persistent_mem_ptr, void *dynamic_mem_ptr);
```

Table 2. hpv_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 hpv_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 (frame by frame).

```
int32_t hpv_setParam(hpv_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 3. hpv_setParam

I/O	Name	Type	Description
Input	input_static_param_ptr	hpv_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 hpv_getParam function

This procedure gets the module's static parameters from the module's 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 (frame by frame).

```
int32_t hpv_getParam(hpv_static_param_t *input_static_param_ptr, void
*persistent_mem_ptr);
```

Table 4. hpv_getParam

I/O	Name	Type	Description
Input	input_static_param_ptr	hpv_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 hpv_setConfig function

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

```
int32_t hpv_setConfig(hpv_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 5. hpv_setConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	hpv_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 hpv_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 hpv_reset() and setParam() routines call).

```
int32_t hpv_getConfig(hpv_dynamic_param_t *input_dynamic_param_ptr, void
*persistent_mem_ptr);
```

Table 6. hpv_getConfig

I/O	Name	Type	Description
Input	input_dynamic_param_ptr	hpv_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 hpv_process function

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

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

Table 7. hpv_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 only for 2.0 to 2.0 processing.

2.2 External definitions and types

In order to facilitate the integration in the main frameworks, some types and definitions have been defined.

2.2.1 Input and output buffers

The HPV 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 used each time before calling to the processing routine; otherwise 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, 6 for 5.1, 8 for 7.1
nb_bytes_per_Sample	int32_t	Dynamic data in number of bytes (2 for 16-bit data, …)
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
HPV_ERROR_NONE	0	OK - No error detected
HPV_BAD_HW	-1	May happen if the library is not used with the right HW
HPV_ERROR_BAD_STRENGHT	-2	HPV strength must be between 0 and 100%
HPV_ERROR_BAD_HEADTRACKING_AZIMUTH	-3	HPV head tracking azimuth must be between -175 and +180 degrees
HPV_ERROR_BAD_AUDIOMODE	-4	Error returned in case of unsupported mode
HPV_ERROR_BAD_BUFFER_LENGTH	-5	Error returned if buffer length is smaller than 2 ms or if not a multiple of 4 samples

2.3 Static parameters structure

Some static parameters must be set before calling the processing routine.

```
struct hpv_static_param {
    int32_t HpvAudioMode;
    int32_t HpvSamplingRate;
    int32_t HpvLfeEnable;
};
typedef struct hpv_static_param hpv_static_param_t;
```

Table 10. Static parameters structure

Name	Type	Description
HpvAudioMode	int32_t	Values taken from eHvpAcInput_Supported enumeration below
HpvSamplingRate	int32_t	I/O sampling rate in Hz
HpvLfeEnable	int32_t	1 if LFE is part of input stream (5.1 or 7.1 inputs), else 0

The possible audio modes are described below:

```
enum eHpvAcInput_Supported
{
   AINPUT_10 = 1,  /* C */
   AINPUT_20 = 2,  /* L, R */
   AINPUT_32 = 7,  /* L, R, C, Ls, Rs */
   AINPUT_34 = 11,  /* L, R, C, Ls, Rs, Csl, Csr */
   ANB_INPUT
};
```

2.4 Dynamic parameters structure

Four dynamic parameters can be used.

```
struct hpv_dynamic_param {
      int32_t HpvEnable;
      int32_t HpvStrength;
      int32_t HpvHTEnable;
      int32_t HpvHTAzimuth;
};
typedef struct hpv_dynamic_param hpv_dynamic_param_t;
```

Table 11. Dynamic parameters structure

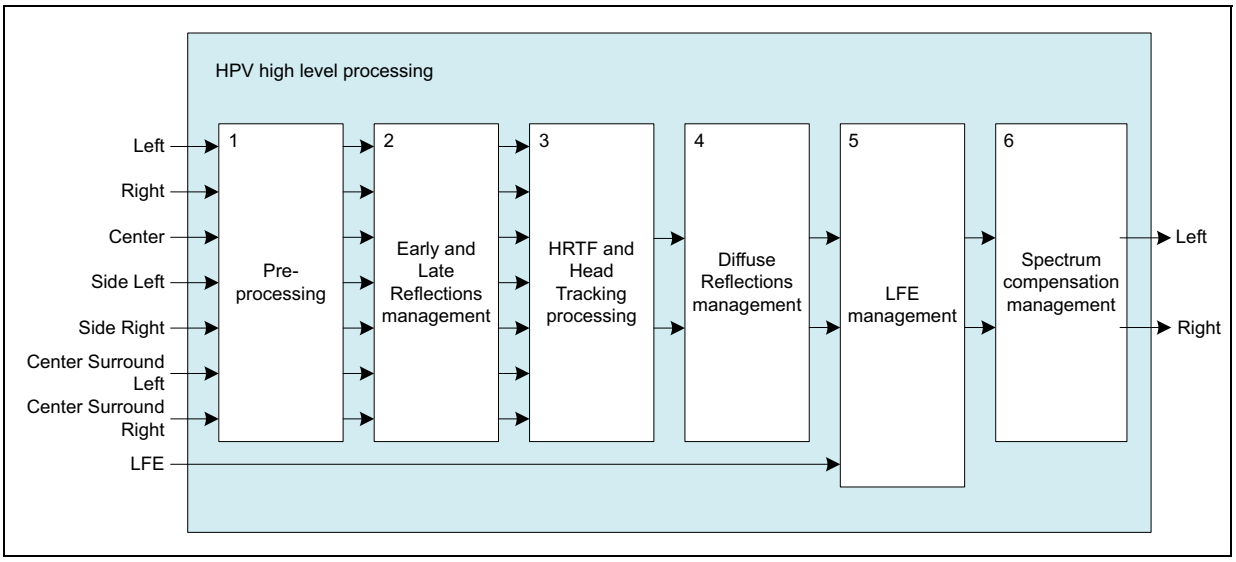
Name	Type	Description
HpvEnable	int32_t	1 to enable the HPV effect, else 0
HpvStrength	int32_t	Effect strength from 0% to 100%
HpvHTEnable	int32_t	1 to activate head tracking usage, else 0
HpvHTAzimuth	int32_t	Head rotation angle (used when head tracking usage is activated). Values are in the range]-180: +180], +90 meaning head has turned right 90 degrees.

3 Algorithm description

3.1 Processing steps

The block diagram of the Hpv module is described in *Figure 2*.

Figure 2. Block diagram of the HPV module



1. **Pre-processing block**: Preparation of the input signal for a better Early Reflections (ER) management by producing some phase diffusion and channel separation.
2. **Early and Late Reflections management block**: Generation of early and late reflections.
3. **HRTF and Head Tracking processing block**: Virtualization from 7 channels to 2 channels based on Head-Related Transfer Functions modeling.
4. **Diffuse Reflections management block**: Generation of diffuse reflections for a better reverberation modeling.
5. **LFE management block**: Omnidirectional LFE channel is combined with virtualized signals.
6. **Spectrum compensation management block**: Signal equalization for a better spectrum preservation.

3.2 Data formats

The module supports fixed point data in Q15 or Q31 format, with a mono, stereo, 5.1 and 7.1 interleaved pattern at 48kHz input sampling frequency.

3.3 Performance assessment

There is no objective measurement available for this module; performances are only based on a subjective assessment.

Below a list of subjective indicators that could be used to evaluate the effect quality:

- **Balance between Left Front and Right Front:** capacity to avoid changing energy on one front channel as compared to the other.
- **Balance between Left Surround and Right Surround**: capacity to avoid changing energy on one surround channel as compared to the other.
- **Center image stability**: ability to keep the center image at the center speaker, or between the left and right front loudspeakers.
- Distinction between front and surround channels.
- **Out of head effect**: Sensation that the sound does not come from the headphones, very close to your ears, but from a farther source. Received sounds should be perceived as less aggressive for ears.
- **Sound source direction**: precision in the virtual sound source (30 degrees for front signals, 110 degrees for side signals and 150 degrees for rear signals).
- **Spectrum preservation**: ability to keep the original spectrum perception, wherever the virtual sound comes from.

4 System requirements and hardware setup

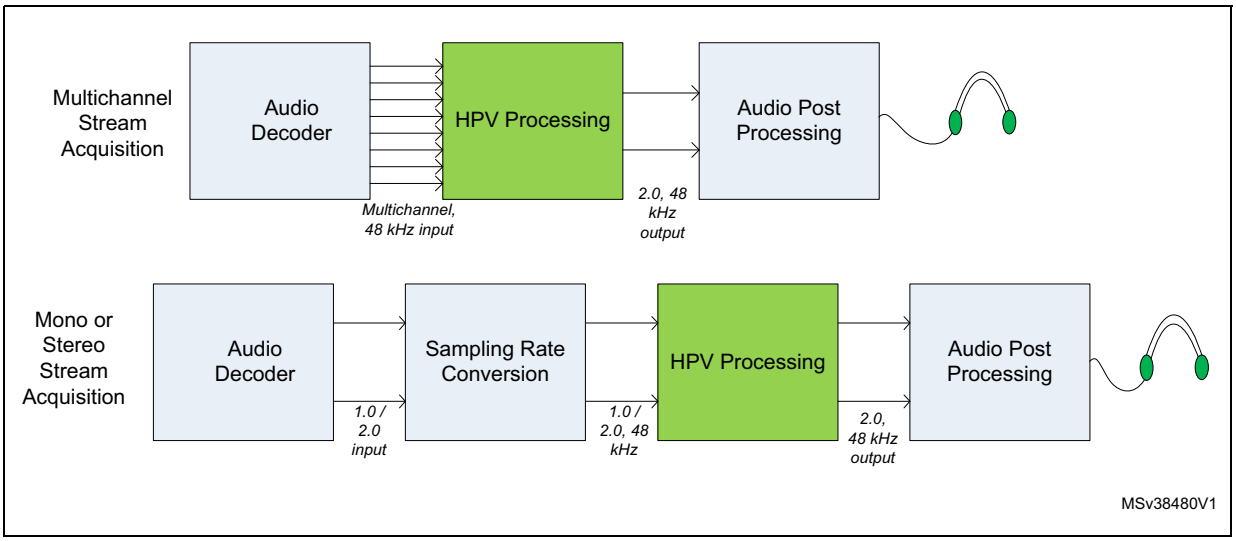
HPV libraries are built to run either on a Cortex M4 or on a Cortex M7 core, without FPU usage. They can be integrated and run on corresponding STM32F4/STM32L4 or STM32F7 family devices. There is no other HW dependency.

4.1 Recommended setup for optimal setup

The library processing should be placed just after the multichannel decoder for the multichannel streams at 48 kHz, or after the sampling rate conversion for stereo streams at any sampling frequencies. Note that only 48 kHz sampling frequency is supported in this SW version.

There is no need for this module to be positioned close to the audio DAC, and some graphical equalizer and volume management modules can be placed after it, without affecting the virtualization perception.

Figure 3. HPV positioning in a basic audio chain

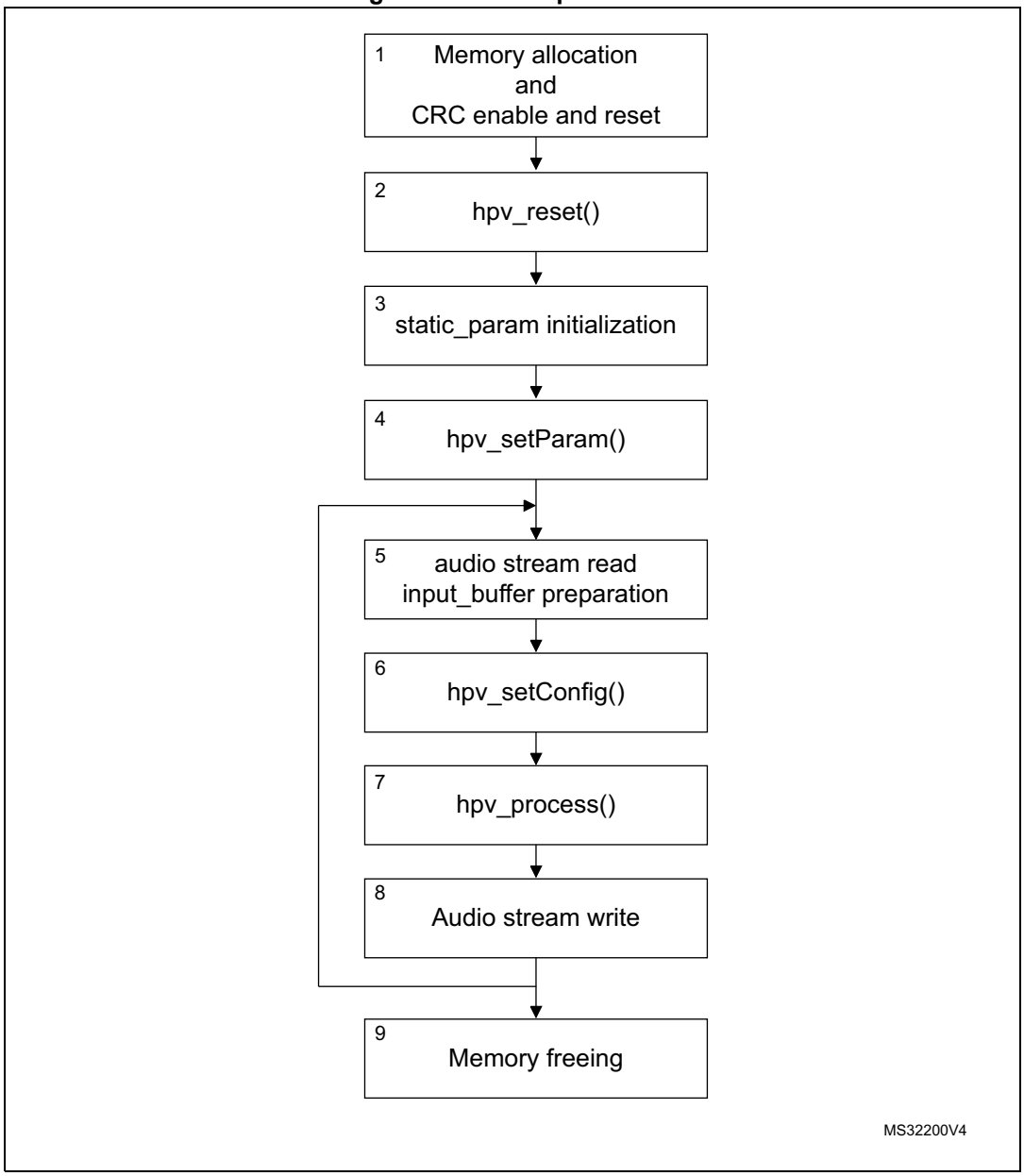


4.1.1 Module integration example

Cube expansion HPV integration examples are provided on STM32746G-Discovery and STM32469I-Discovery boards. Please refer to provided integration code for more details.

4.1.2 Module integration summary

Figure 4. API call procedure



1. As explained above, the module' s persistent and scratch memories have to be allocated, as well as the input and output buffer, according to the structures defined in *Section 2.2.1: Input and output buffers*. Furthermore, as HPV library run on STM32 devices, CRC HW block must be enable and reset.
2. Once the memory is allocated, the call to hpv_reset() function initializes the internal variables.
3. The module's static configuration can now be set by initializing the static_param structure, once the audio mode is known.
4. Call the hpv_setParam() routine to send the static parameters from the audio framework to the module.
5. The audio stream is read from the proper interface and the input_buffer structure has to be filled in according to the stream characteristics (number of channels, sample rate, interleaving and data pointer). The output buffer structure has to be set as well.
6. Get the dynamic parameters when they are updated and call the hpv_setConfig() routine to send the

dynamic parameters from the audio framework to the module.

7. Call the main processing routine to apply the effect.
8. The output audio stream can now be written in the proper interface.
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 an audio framework to play stereo samples at 48kHz, user launches a player and the output file will be decoded and played with virtualization effect on two physical output headphones.

The *HPVEnable* is used to enable and disable the effect.

The *HPVStrength* dynamic parameter is provided to change virtualization effect strength. “Out of the head” effect and reverberation can be attenuated using this parameter.

The *HpvHTEnable* dynamic parameter is used to enable the head tracking feature. When it is activated, HpvHTAzimuth parameter is taken into account.

The *HpvHTAzimuth* dynamic parameter should contain the relative azimuth of the head compared to the sound coming from the front of the listener. A typical usage of this feature is gaming, connecting sensors (accelerometer, gyroscope) output to *HpvHTAzimuth* input. If head tracking feature is activated, sound will come from fixed sources, whatever the head rotation around vertical axe.

6 Revision history

Table 12. Document revision history

Date	Revision	Changes
17-Feb-2016	1	Initial release.
21-Mar-2017	2	Updated: – *Table 1: Resources summary*, *Table 2: hpv_reset*, *Table 3: hpv_setParam*, *Table 4: hpv_getParam*, *Table 5: hpv_setConfig*, *Table 6: hpv_getConfig*, *Table 7: hpv_process*, *Table 8: Input and output buffers*, *Table 11: Dynamic parameters structure* – *Section 1.2: Module configuration*, *Section 1.3: Resources summary*, *Section 2: Module Interfaces*, *Section 2.1.1: hpv_reset function*, *Section 2.1.2: hpv_setParam function*, *Section 2.1.3: hpv_getParam function*, *Section 2.1.4: hpv_setConfig function*, *Section 2.1.5: hpv_getConfig function*, *Section 2.1.6: hpv_process function*, *Section 4.1.1: Module integration example*, *Section 4.1.2: Module integration summary*, *Section 5: How to run and tune the application*
08-Jan-2018	3	Replace RPNs X-CUBE-AUDIO-F4, X-CUBE-AUDIO-F7 and X-CUBE-AUDIO-L4 with X-CUBE-AUDIO.