Introduction

The Audio Engine post-processing on STM32F4xx - Omni2 multichannel library user manual describes the software interface and requirements of the Omni2 multichannel module. It has been designed for the audio application developers who integrate this module into a main program. It provides a rough understanding of the underlying algorithm.

The Omni2 multichannel library implements the multichannel audio virtualization from mono to 7.1 input signals, including the stereo widening effect for stereo inputs.

This library is part of the STM32-AUDIO100A firmware package.
# Contents

## 1 Module overview

1.1 Algorithm function ......................................... 6  
1.2 Module configuration ....................................... 8  
1.3 Resources summary ........................................ 8

## 2 Module Interfaces ......................................... 10

2.1 APIs .............................................................. 10  
2.1.1 omni2_reset function .................................... 10  
2.1.2 omni2_setParam function ................................ 10  
2.1.3 omni2_getParam function ................................ 11  
2.1.4 omni2_setConfig function ............................... 11  
2.1.5 omni2_getConfig function ................................ 12  
2.1.6 omni2_process function .................................. 12  
2.2 External definitions and types ............................. 12  
2.2.1 Input and output buffers ................................ 12  
2.2.2 Returned error values .................................. 13  
2.3 Static parameters structure ................................. 13  
2.4 Dynamic parameters structure .............................. 14

## 3 Algorithm description ....................................... 16

3.1 Processing steps ............................................. 16  
3.2 Data formats .................................................... 16  
3.3 Performance assessment ....................................... 16

## 4 Application description ..................................... 18

4.1 System requirements and hardware setup .................. 18  
4.1.1 Recommended setup for stereo widening effect  
(Omni2DownFiringSpeakers not set) .......................... 18  
4.1.2 Recommended setup for stereo widening effect for closely spaced speakers (Omni2DownFiringSpeakers set) .............. 19  
4.1.3 Recommended setup for multichannel virtualizer effect  ........................................................................ 20  
4.2 Recommendations for an optimal setup ................. 20  
4.2.1 Memory allocation ........................................ 21  
4.2.2 Module API calls .......................................... 21
List of tables

Table 1. Sampling rates ................................................................. 6
Table 2. Resources summary ......................................................... 8
Table 3. Frequency requirements (MHz) ........................................... 9
Table 4. omni2_reset ................................................................. 10
Table 5. omni2_setParam ............................................................. 11
Table 6. omni2_getParam .............................................................. 11
Table 7. omni2_setConfig ............................................................ 11
Table 8. omni2_getConfig ............................................................. 12
Table 9. omni2_process ................................................................. 12
Table 10. Input and output buffers ................................................ 13
Table 11. Returned error values ..................................................... 13
Table 12. Static parameters structure ........................................... 14
Table 13. Dynamic parameters structure ....................................... 15
Table 14. Recommended angle (Omni2DownFiringSpeakers not set) .... 18
Table 15. Setup examples on the speaker distance (Omni2DownFiringSpeakers not set) 18
Table 16. Recommended angle (Omni2DownFiringSpeakers set) .......... 19
Table 17. Setup examples on the speaker distance (Omni2DownFiringSpeakers set) 19
Table 18. Virtualized sound compliant with ITU-T 7.1 speaker layout standards 20
Table 19. Document revision history .............................................. 26
List of figures

Figure 1. Mono to stereo perception for mono inputs ............................................. 7
Figure 2. Stereo widening perception for stereo inputs ........................................ 7
Figure 3. Virtualizer perception (up to 7.1 channels) ........................................... 8
Figure 4. Block diagram of the Omni2 module ..................................................... 16
Figure 5. Setup for stereo widening effect (Omni2DownFiringSpeakers not set) ........ 18
Figure 6. Setup for stereo widening effect (Omni2DownFiringSpeakers set) .......... 19
Figure 7. API call procedure ............................................................................. 23
1 Module overview

1.1 Algorithm function

The module provides functions to handle mono to stereo expansion, stereo widening and multichannel audio virtualization depending on the used library.

Table 1 describes the supported sampling rates and the input/output formats depending on which library is used:

<table>
<thead>
<tr>
<th>Library</th>
<th>Audio effect</th>
<th>Channel conversions</th>
<th>Supported sampling frequencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>lib_omni2_multichannel_m4</td>
<td>Mono2Stereo</td>
<td>1.0 to 2.0</td>
<td>32, 44.1 and 48 kHz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1.0 to 3.0 (2.0 + center)</td>
<td></td>
</tr>
<tr>
<td>Std_omni2_multichannel_m4</td>
<td>Stereo Widening</td>
<td>2.x to 2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.x to 3.0 (2.0 + center)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.x to 2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>3.x to 3.0 (2.0 + center)</td>
<td></td>
</tr>
<tr>
<td>Std_omni2_multichannel_m4</td>
<td>Stereo Widening for matrix encoded input</td>
<td>2.xt to 2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>2.xt to 3.0 (2.0 + center)</td>
<td></td>
</tr>
<tr>
<td>Std_omni2_multichannel_m4</td>
<td>Multichannel Virtualization</td>
<td>5.x to 2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>5.x to 3.0 (2.0 + center)</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.x to 2.0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>7.x to 3.0 (2.0 + center)</td>
<td></td>
</tr>
</tbody>
</table>

The module is fully functional and validated with the configurations in green in Table 1 above.

The figures below present the effect perception with only two physical loudspeakers.
Figure 1. Mono to stereo perception for mono inputs

Sound image has a much larger size if left and right channels are decorrelated

Actual physical speakers

Sound image has a signal spot for mono signal

Figure 2. Stereo widening perception for stereo inputs

Actual physical speakers

Perceived virtual speakers

Perceived virtual speakers
The virtualizer perception (up to 7.1 channels) gives the listener the impression of a multi-speaker sensation with stereo speakers.

1.2 Module configuration

The module supports mono and multichannel interleaved 16-bit and 32-bit I/O data up to 7.1 format.

lib_omni2_multichannel_m4.a and lib_omni2_multichannel_32b_m4.a libraries should be used for the multichannel virtualization use case, even if it also supports stereo widening effect.

1.3 Resources summary

Table 2 contains the module requirements for the Flash, stack and RAM memories.

The required core frequencies (in MHz) are estimated using EWARM v6.50 profiler, while in parenthesis values have been measured on real hardware, running on the STM32F407IG chipset.

<table>
<thead>
<tr>
<th>Omni2</th>
<th>Flash code stereo (.text)</th>
<th>Flash data (.rodata)</th>
<th>Stack</th>
<th>Static RAM</th>
<th>Dynamic RAM</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omni2 for multichannel virtualization</td>
<td>4306 Bytes</td>
<td>504 Bytes</td>
<td>310 Bytes</td>
<td>3028 Bytes</td>
<td>1152 Bytes</td>
</tr>
<tr>
<td>Omni2 for multichannel virtualization, 32 bits I/O</td>
<td>4260 Bytes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 3 contains module frequency requirements.
The required core frequencies (in MHz) are estimated using EWARM v6.50 profiler, with 10 ms buffers at 48 kHz, while in parenthesis values have been measured on real hardware, running on the STM32F407IG chipset.

<table>
<thead>
<tr>
<th>Table 3. Frequency requirements (MHz)</th>
<th>Stereo widening (MHz)</th>
<th>Stereo widening 32 bits (MHz)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Closely spaced speakers</td>
<td>5.4 (5.2)</td>
<td>4.8 (4.5)</td>
</tr>
<tr>
<td>Mono2Stereo</td>
<td>7.7</td>
<td>7.1</td>
</tr>
<tr>
<td>Stereo in / Stereo out</td>
<td>23.5 (23.5)</td>
<td>23 (23)</td>
</tr>
<tr>
<td>Stereo in / Stereo + Center out</td>
<td>23.9 (23.9)</td>
<td>24.1 (24)</td>
</tr>
<tr>
<td>5.1 virtualization</td>
<td>26.2 (26)</td>
<td>25.6 (25.2)</td>
</tr>
<tr>
<td>7.1 virtualization</td>
<td>26.6 (26.2)</td>
<td>26 (25.6)</td>
</tr>
</tbody>
</table>
2 Module Interfaces

Two files are needed to integrate this module: lib_omni2_multichannel_m4.a or lib_omni2_multichannel_32b_m4.a library, and the omni2_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: omni2_reset, omni2_setParam, omni2_getParam, omni2_setConfig, omni2_getConfig, and omni2_process.

2.1.1 omni2_reset function

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

```c
int32_t omni2_reset(void *static_mem_ptr, void *dynamic_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Input</td>
<td>dynamic_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal dynamic memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>–</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

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

2.1.2 omni2_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 the values which cannot be changed during the module processing (frame by frame).

```c
int32_t omni2_setParam(omni2_static_param_t *input_static_param_ptr, void *static_mem_ptr);
```
2.1.3 omni2_getParam function

This procedure gets the module 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).

```c
int32_t omni2_getParam(omni2_static_param_t *input_static_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_static_param_ptr</td>
<td>omni2_static_param_t*</td>
<td>Pointer to static parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>–</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

2.1.4 omni2_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 the module processing (after the reset and setParam routines).

```c
int32_t omni2_setConfig(omni2_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_dynamic_param_ptr</td>
<td>omni2_dynamic_param_t*</td>
<td>Pointer to dynamic parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td>–</td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>
2.1.5 omni2_getConfig function

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

```
int32_t omni2_getConfig(omni2_dynamic_param_t *input_dynamic_param_ptr, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_dynamic_param_ptr</td>
<td>omni2_dynamic_param_t *</td>
<td>Pointer to dynamic parameters structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td></td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

2.1.6 omni2_process function

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

```
int32_t omni2_process(buffer_t *input_buffer, buffer_t *output_buffer, void *static_mem_ptr);
```

<table>
<thead>
<tr>
<th>I/O</th>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>input_buffer</td>
<td>buffer_t *</td>
<td>Pointer to input buffer structure</td>
</tr>
<tr>
<td>Output</td>
<td>output_buffer</td>
<td>buffer_t *</td>
<td>Pointer to output buffer structure</td>
</tr>
<tr>
<td>Input</td>
<td>static_mem_ptr</td>
<td>void *</td>
<td>Pointer to internal static memory</td>
</tr>
<tr>
<td>Returned value</td>
<td></td>
<td>int32_t</td>
<td>Error value</td>
</tr>
</tbody>
</table>

This process routine can run in place only in case of w.x to y.z with (w+x) ≥ (y+z). For instance, processing such as the stereo widening effect (2.0 to 2.0) or 5.1 virtualization effect (5.1 to 2.0) can run in place.

2.2 External definitions and types

For genericity reasons and to facilitate the integration in the main frameworks, some types and definitions have been defined.

2.2.1 Input and output buffers

The 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 10. Input and output buffers

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>nb_channels</td>
<td>int32_t</td>
<td>Number of channels in data: 1 for mono, 2 for stereo</td>
</tr>
<tr>
<td>nb_bytes_per_sample</td>
<td>int32_t</td>
<td>Dynamic data in number of bytes (2 for 16-bit data, …)</td>
</tr>
<tr>
<td>data_ptr</td>
<td>void *</td>
<td>Pointer to data buffer (must be allocated by the main framework)</td>
</tr>
<tr>
<td>buffer_size</td>
<td>int32_t</td>
<td>Number of samples per channel in the data buffer</td>
</tr>
<tr>
<td>mode</td>
<td>int32_t</td>
<td>Buffer mode: 0 = not interleaved, 1 = interleaved</td>
</tr>
</tbody>
</table>

2.2.2 Returned error values

Possible returned error values are described below:

Table 11. Returned error values

<table>
<thead>
<tr>
<th>Definition</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>OMNI2_ERROR_NONE</td>
<td>0</td>
<td>OK - No error detected</td>
</tr>
<tr>
<td>OMNI2_ERROR</td>
<td>-1</td>
<td>Could be a bad sampling frequency, or a bad dynamic memory allocation</td>
</tr>
<tr>
<td>OMNI2_ERROR_PARSE_COMMAND</td>
<td>-2</td>
<td>Internal error - covers bad internal settings</td>
</tr>
<tr>
<td>OMNI2_BAD_HW</td>
<td>-3</td>
<td>May happen if the library is not used with the right hardware</td>
</tr>
</tbody>
</table>

2.3 Static parameters structure

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

```c
struct omni2_static_param {
    int32_t Omni2CentreOutput;
    int32_t AudioMode;
    int32_t SamplingFreq;
};
```

typedef struct omni2_static_param omni2_static_param_t;
The audio modes available are described below:

```c
enum eAcMode_Supported
{
    AMODE20t = 0x0,/* Stereo channels for dolby pro logic */
    AMODE10 = 0x1,/* Mono channel (1.0) */
    AMODE20 = 0x2,/* Stereo channels (2.0) */
    AMODE30 = 0x3,/* Stereo + Center channel (3.0) */
    AMODE32 = 0x7,/* Stereo + Center channel + Surround Channels (5.0) */
    AMODE34 = 0xB,/* Stereo + Center channel + Surround Channels + Center Surround Channels (7.0) */
    AMODE20t_LFE = 0x80,/* Stereo channels for dolby pro logic + LFE channel */
    AMODE20_LFE = 0x82,/* Stereo + LFE channel (2.1) */
    AMODE30_LFE = 0x83,/* Stereo + Center channel + LFE channel (3.1) */
    AMODE32_LFE = 0x87,/* Stereo + Center channel + LFE channel + Surround Channels (5.1) */
    AMODE34_LFE = 0x8B,/* Stereo + Center channel + LFE channel + Surround Channels + Center Surround Channels (7.1) */
    AMODE_ID = 0xFF/* End of supported configurations */
};
```

### 2.4 Dynamic parameters structure

Three dynamic parameters can be used.

```c
struct omni2_dynamic_param {
    int32_t   Omni2Enable;
    int32_t   Omni2Strength;
    int32_t   Omni2DownFiringSpeakers;
};

typedef struct omni2_dynamic_param omni2_dynamic_param_t;
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Omni2Enable</td>
<td>int32_t</td>
<td>1 to enable the effect, 0 to disable the effect</td>
</tr>
<tr>
<td>Omni2Strength</td>
<td>int32_t</td>
<td>Used to widen front signals for multichannel and stereo widening modes. The value is from 0% (no widening perception) to 100% (maximum widening perception)</td>
</tr>
<tr>
<td>Omni2DownFiringSpeakers</td>
<td>int32_t</td>
<td>1 to enable the effect if the speakers are closely spaced (~15 degrees listening angle), 0 in case of normal mode (~30 degrees listening angle) for a stereo use case.</td>
</tr>
</tbody>
</table>
3 Algorithm description

3.1 Processing steps

The block diagram of the Omni2 module is described in Figure 4.

Figure 4. Block diagram of the Omni2 module

Routing block: Carries out a premixing of the channels so that it can be processed with a single virtualization structure.

Virtualization block: Applies the HRTF and Crosstalk cancellation function.

Speaker adjustment filter: Processes the audio signal after virtualization processing for speaker rendering and spectrum preservation.

Mono-to-stereo block: Carries out a mono-to-stereo expansion and bypasses the speaker rendering block.

3.2 Data formats

The module supports fixed point data in Q15 or Q31 format, with a mono or a multichannel up to 7.1 interleaved pattern.

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 not to change energy on one front channel as compared to the other.
- **Balance between Left Surround and Right Surround**: capacity not to change energy on one surround channel as compared to the other.
- **Stereo Widening**: ability to increase the audio perception angle to widen the stereo signal.
- Distinction between front and side surround channels
- **Center image stability**: ability to keep the center image at the center loudspeaker, or between the left and right front loudspeakers.
- **Sensitivity to sweet spot**: ability to feel a widening or surround effect moving away from the sweet spot described in Section 4.1.1 and Section 4.1.2.
- **Spectrum preservation**: ability to keep the original spectrum perception, wherever the virtual sound comes from.

*Note:* For more information on the performance, refer to Section 4.1: System requirements and hardware setup and Section 5: How to run and tune the application.
4 Application description

4.1 System requirements and hardware setup

The library is built to run on a Cortex M4 core without FPU usage. It was integrated and validated on some STM32F4 family devices.

4.1.1 Recommended setup for stereo widening effect (Omni2DownFiringSpeakers not set)

The stereo widening effect is designed for a 30 degrees listening angle, that is +/- 15 degrees spacing, which corresponds to a typical TV watching use case.

Figure 5. Setup for stereo widening effect (Omni2DownFiringSpeakers not set)

Table 14. Recommended angle (Omni2DownFiringSpeakers not set)

<table>
<thead>
<tr>
<th>Minimum angle</th>
<th>Typical angle</th>
<th>Maximum angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>20°</td>
<td>30°</td>
<td>40°</td>
</tr>
</tbody>
</table>

Table 15 presents some setup examples and direct impacts on the speaker distance to get a typical listening angle and an optimal stereo widening perception.

Table 15. Setup examples on the speaker distance (Omni2DownFiringSpeakers not set)

<table>
<thead>
<tr>
<th>Inter speaker distance</th>
<th>Speaker/listener distance</th>
<th>Equivalent listening angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.2 m</td>
<td>2.5 m</td>
<td>27°</td>
</tr>
<tr>
<td>1 m</td>
<td>2 m</td>
<td>28°</td>
</tr>
<tr>
<td>0.8 m</td>
<td>1.5 m</td>
<td>30°</td>
</tr>
<tr>
<td>0.6 m</td>
<td>1.0 m</td>
<td>32°</td>
</tr>
</tbody>
</table>
Note: In order to benefit from the stereo widening effect, the listener must be well centered between the two loudspeakers because this effect is very sensitive to the lateral sweet spot. The position of the virtual front channels varies from ± 15° to ± 50°.

4.1.2 Recommended setup for stereo widening effect for closely spaced speakers (Omni2DownFiringSpeakers set)

This mode should be used when the listening angle goes below 20 degrees, that is +/- 10 degrees spacing, which corresponds to a typical TV watching with down firing speakers, or docking stations usage.

Figure 6. Setup for stereo widening effect (Omni2DownFiringSpeakers set)

Table 16. Recommended angle (Omni2DownFiringSpeakers set)

<table>
<thead>
<tr>
<th>Minimum angle</th>
<th>Typical angle</th>
<th>Maximum angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>10°</td>
<td>15°</td>
<td>20°</td>
</tr>
</tbody>
</table>

Table 17 presents some setup examples and direct impacts on the speaker distance to get a typical listening angle and an optimal stereo widening perception.

Table 17. Setup examples on the speaker distance (Omni2DownFiringSpeakers set)

<table>
<thead>
<tr>
<th>Inter speaker distance</th>
<th>Speaker/listener distance</th>
<th>Equivalent listening angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.6 m</td>
<td>2.2 m</td>
<td>15°</td>
</tr>
<tr>
<td>0.3 m</td>
<td>1.7 m</td>
<td>10°</td>
</tr>
<tr>
<td>0.3 m</td>
<td>1.1 m</td>
<td>15°</td>
</tr>
</tbody>
</table>

Note: In order to benefit from the stereo widening effect, the listener must be well centered between the two loudspeakers because this effect is very sensitive to the lateral sweet spot. The position of the virtual front channels varies from ± 10° to ± 30°.
4.1.3 **Recommended setup for multichannel virtualizer effect**

Like the Stereo Widening effect, the multichannel virtualizer effect has been designed for a 30 degrees listening angle, that is +/- 15 degrees spacing, which corresponds to a typical TV watching use case.

Table 18. Virtualized sound compliant with ITU-T 7.1 speaker layout standards

![Diagram showing virtual speakers and physical speakers with angles labeled: 30°, 110°, 150°, and 150°.]

i.e. Front channels are at ± 30°, Side channels are at ± 110° and Rear channels are at ± 150°.

In stereo widening mode, Front channels should be at ± 45°.

4.2 **Recommendations for an optimal setup**

The library processing should be placed just after the multichannel decoder and before the sampling rate conversion in order to process the audio signal at 32, 44.1 or 48 kHz, and avoid doing sampling rate conversions on multichannel. There is no need for this module to be close to the audio DAC, and some graphical equalizer and volume management modules can be placed after it without affecting the widening or virtualization perception.

To integrate this module in an audio software framework, follow the next steps.
4.2.1 Memory allocation

First of all, all the memory used by the module must be allocated.

The static parameters structure and dynamic parameters structure are exported in the omni2_glo.h file, so that the memory for these structures can be allocated as written in the example below:

```c
/* Omni2 static and dynamic parameters structures memory allocation */
omni2_static_param_t *static_param_ptr = malloc(sizeof(omni2_static_param_t));
omni2_dynamic_param_t *dynamic_param_ptr = malloc(sizeof(omni2_dynamic_param_t));
```

Where:

- `static_param_ptr` pointer is used by omni2_setParam() and omni2_getParam() routines.
- `dynamic_param_ptr` pointer is used by omni2_setConfig() and omni2_getConfig() routines.

Next, the static and dynamic memory required by the module must be allocated by the framework. Structures are hidden to the audio framework, but their sizes are exported as a constant in the `omni2_glo.h` file, so that the memory allocation can be done as written below:

```c
/* Omni2 memory structure memory allocation */
void *static_mem_ptr = malloc(omni2_static_mem_size);
void *dynamic_mem_ptr = malloc(omni2_dynamic_mem_size);
```

Where:

- `dynamic_mem_ptr` pointer is a parameter of the omni2_reset() routine
- `static_mem_ptr` pointer is a parameter of all exported APIs.

Then, it is necessary to allocate the memory for the input and output audio buffers.

4.2.2 Module API calls

Once the memory has been allocated, the omni2_reset() routine must be called to initialize the module's static memory. The omni2_reset() routine must be called each time the audio processing has been stopped and started.

The omni2_getParam() routine can be called while the run time process has not started to extract the current static parameters used, if needed.

The static parameters can be set as soon as the input sampling frequency and the stream information are known. The omni2_setParam() routine must be called to configure the module's internal memory with the corresponding sampling frequencies and the audio mode as in the example below:

```c
/* Omni2 static parameters setting */
omni2_static_param.Omni2CentreOutput = 0; /* center output disabled */
omni2_static_param.SamplingFreq = 48000; /* I/O sampling frequency */
omni2_static_param.AudioMode = AMODE32_LFE; /* 5.1 -> 2.0 : 5.1 virtualization */
error = omni2_setParam(&omni2_static_param, omni2_static_mem_ptr);
```
Now that the hardware has been configured and the module has been initialized and configured, the run time processing can start.

Every time the dynamic parameters change, they are taken into account by the module thanks to the `omni2_setConfig()` routine:

```c
/* Omni2 dynamic parameters setting */
omni2_dynamic_param.Omni2Enable = Omni2CtrlStruct->Omni2Enable;     /* 0 = disable, 1 = enable */
omni2_dynamic_param.Omni2Strength = Omni2CtrlStruct->Omni2Strength;    /* value from 0 to 100 */
omni2_dynamic_param.Omni2DownFiringSpeakers = Omni2CtrlStruct->Omni2DownFiringSpeakers;    /* 0 = disable, 1 = enable */
error = omni2_setConfig(&omni2_dynamic_param, omni2_static_mem_ptr);
```

At each new frame, the input buffer structure fields must be filled in as in the example below, as well as the data address for the output buffer structure. Then, the `omni2_process()` routine can be called:

```c
/* Omni2 input buffer configuration and processing all */
input_buffer_t.data_ptr = input_buffer_ptr;
input_buffer_t.buffer_size = input_buffer_size;
input_buffer_t.mode = INTERLEAVED;
input_buffer_t.nb_bytes_per_Sample = 2;
input_buffer_t.nb_channels = 6;
output_buffer_t.data_ptr = output_buffer_ptr;
output_buffer_t.buffer_size = input_buffer_size;
output_buffer_t.mode = INTERLEAVED;
output_buffer_t.nb_bytes_per_Sample = 2;
output_buffer_t.nb_channels = 2;
error = omni2_process(&omni2_input_buffer_t, &omni2_output_buffer_t, omni2_static_mem_ptr);
```
4.2.3 Module integration summary

Figure 7. API call procedure

1. Memory allocation
2. omni2_reset()
3. static_param initialization
4. omni2_setParam()
5. audio stream read
   input_buffer preparation
6. omni2_setConfig()
7. omni2_process()
8. Audio stream write
9. Memory freeing

1. As explained above, the module’s static and dynamic structures 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.

2. Once the memory has been allocated, the call to omni2_reset() function initializes the internal variables.

3. The module's static configuration can now be set by initializing the static_param structure, once the input sampling frequency and the audio mode are known.

4. Call the omni2_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 omni2_setConfig() routine to send the dynamic parameters from the audio framework to the module.

7. Call the processing main 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 has been integrated into an 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 no sampling rate conversion is available.

The Omni2Enable from the modules's dynamic parameters is used to enable and disable the effect.

The Omni2DownFiringSpeakers field allows to switch from largely speakers mode (sound bars UC) to closely speakers mode (small sound bars or docking stations).

The Omni2Strength field is used to change the virtual listening angle. 0% means that virtual speakers are close to physical speakers, while 100% leads to the widest virtual angle (typically 100 degrees). It is recommended to have Omni2Strength value between 60% and 80% for largely spaced speakers, and keep 100% for closely spaced speakers (that is when Omni2DownFiringSpeakers is activated).
# Revision history

Table 19. Document revision history

<table>
<thead>
<tr>
<th>Date</th>
<th>Revision</th>
<th>Changes</th>
</tr>
</thead>
<tbody>
<tr>
<td>26-Aug-2013</td>
<td>1</td>
<td>Initial release.</td>
</tr>
<tr>
<td>28-Nov-2014</td>
<td>2</td>
<td>Updated RPN on cover page</td>
</tr>
<tr>
<td>10-Dec-2014</td>
<td>3</td>
<td>Updated Section 5.</td>
</tr>
</tbody>
</table>
IMPORTANT NOTICE – PLEASE READ CAREFULLY

STMicroelectronics NV and its subsidiaries ("ST") reserve the right to make changes, corrections, enhancements, modifications, and improvements to ST products and/or to this document at any time without notice. Purchasers should obtain the latest relevant information on ST products before placing orders. ST products are sold pursuant to ST's terms and conditions of sale in place at the time of order acknowledgement.

Purchasers are solely responsible for the choice, selection, and use of ST products and ST assumes no liability for application assistance or the design of Purchasers' products.

No license, express or implied, to any intellectual property right is granted by ST herein.

Resale of ST products with provisions different from the information set forth herein shall void any warranty granted by ST for such product.

ST and the ST logo are trademarks of ST. All other product or service names are the property of their respective owners.

Information in this document supersedes and replaces information previously supplied in any prior versions of this document.

© 2014 STMicroelectronics – All rights reserved