# AN4309
# Application note

# Interfacing an STM32L1xx microcontroller with an external I2S audio codec to play audio files

## Introduction

This application note describes how to use the STM32L1xx $I^2S$ feature to play audio files using an external codec (reference STSW-STM32135).

The $I^2S$ protocol is widely used to transfer audio data from a microcontroller/DSP to an audio codec in order to play melodies (stored in a memory) or, to capture analog sound (from a microphone).

The STM32L1xx allows $I^2S$ audio communication using the SPI peripheral, and implements specific functionalities for this communications mode.

The first section of this application note can be skipped by advanced users.

*Note: Throughout this document, and unless otherwise specified, the term of $I^2S$ will be used to refer to the $I^2S$ feature of the SPI peripheral that is implemented in STM32L1xx microcontrollers.*

# Contents

# 1 I²S general description

## 1.1 I²S protocol

I²S (IC-to-IC sound) is an audio data transfer standard using a three-line bus for serial and synchronous data transmission.

Data are transmitted on the SD line (Serial Data) in Little Endian format (MSB first). Data length is not limited (usually 16/20/24/32 bits). Data are synchronized by the rising or falling edge of SCK (Serial Clock) for the transmitter, and by the falling edge of SCK for the receiver. Refer to *Figure 1*.

Data represent stereo digital sound, so each sample contains two words, the right-channel sample and the left-channel sample. Instead of using two data channels, multiplexing is performed by transmitting each word over half a sampling period, which doubles the sampling rate, and makes it possible to transmit two words per period.

A control signal WS (Word Select) is then used to determine if the word being sent is the right or left one. This signal also determines the beginning and the end of the data: there is no need to fix the data length. Receiver and transmitter data lengths can therefore be different, as well as the right and left data lengths.

WS is synchronized to either the rising or the falling edge of SCK and precedes the MSB by one SCK period in order to have enough time to store and shift operations.

As in most communication protocols, there must be a master and a slave. The master provides and controls the SCK clock and the WS signal, while the slave only sends or receives data. The master can be the receiver, the transmitter or a third element (Controller). Refer to *Figure 2*.

**Figure 1. I²S Phillips protocol waveforms 16/32-bit**

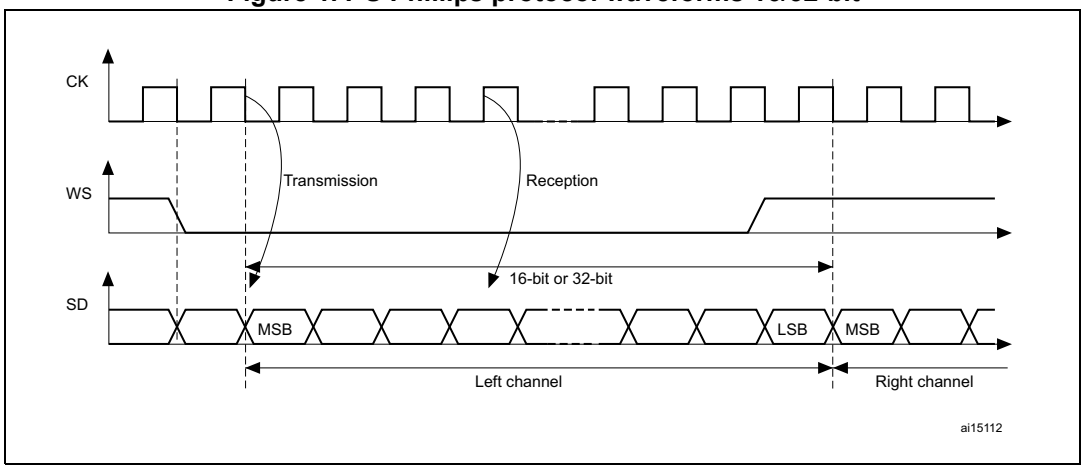
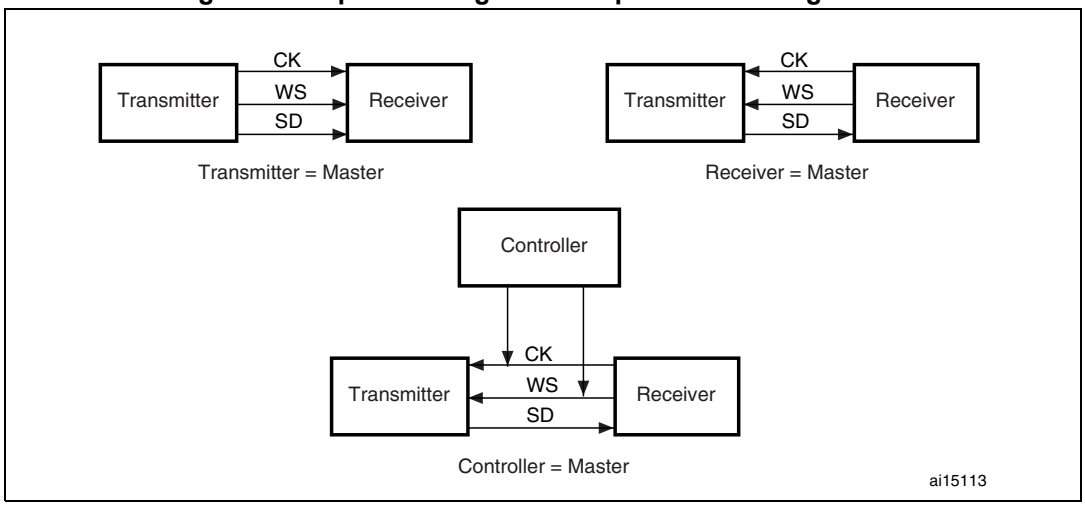


**Figure 2. I²S protocol signal description and configuration**




## 1.2 STM32L1xx I²S feature presentation

The STM32L1xx implements the I²S feature as a mode included in the SPI peripheral. The user must choose either the SPI mode or the I²S mode (software configuration).

The STM32L1xx I²S is available in simplex mode only (receive-only or transmit-only), the communication direction is configured by software.

The I²S peripheral supports four audio protocols (configurable by software):

- I²S Phillips protocol
- MSB protocol
- LSB protocol
- PCM protocol (including PCM Short and PCM Long)

It also supports most audio frequencies (8 kHz, 16 kHz, 22.05 kHz, 32 kHz, 44.1 kHz, 48 kHz, etc.)

The data format is programmable to 16-, 24- or 32-bit data length (for each channel), MSB first, and to 16- or 32-bit packet length (for each channel).

The WS signal assignment is managed by hardware and a relative flag (CHSIDE) is available to monitor the channel side (for Phillips, MSB and LSB standards).

The I²S peripheral can be configured as the master or the slave in the audio communication. The I²S generates its own clock (independent of the SPI clock used to interface registers to the APB bus) using a 9-bit prescaler and designed to reach accurate audio frequencies (8 kHz, 16 kHz, 22.05 kHz, 44.1 kHz, 32 kHz, 48 kHz, etc.)[(a)]. When configured in master mode, the peripheral is able to output an additional master clock (MCLK) at a fixed rate: 256 × $F_S$ (where $F_S$ is the audio frequency).

a. The sampling frequency is the bit clock frequency (CK) and is equal to:
$F_S$ × number of bits per channel × number of channels, where $F_S$ is the WS frequency in the Phillips, MSB and LSB standards and, the WS/2 frequency in PCM mode.

To decide if MCLK should be generated or not, the following facts have to be taken into consideration:

- The external I²S device requirements (codec/DAC).
  In general these devices need a master clock (usually at the rate 256 × $F_S$) to perform internal and sampling operations.
- The audio frequency accuracy is, in some cases, compromised by enabling the MCLK output.

The audio communication can be controlled in one of the following ways:

  – By polling on the TXE/RXNE flag (bits 1/0 in SPI_CR2 register): wait until TXE/RXNE flag is set then write/read the channel wave data to/from the SPI_DR register. (Suitable for tests/small applications, etc.)
  – Interrupt on TXE/RXNE: configure and enable the transmit/receive interrupt. And in the interrupt subroutine, write/read the channel wave data to/from the SPI_DR register. (Suitable for most applications/RT software, etc.)
  – DMA transfer: configure the DMA to load/unload the data from/to the SPI_DR register on each Rx/Tx request. (Suitable for high-performance requirements.)

*Note: In I²S mode, the DMA is used in exactly the same way as the SPI mode (with respect to the supported audio transmission protocols, the CRC feature is not available in I²S mode).*

The choice of the SYSCLK frequency directly impacts the I²S transmission quality (in master mode): the sampling clock (CK) and WS clock are derived directly from SYSCLK divided by a 9-bit prescaler in order to obtain the most accurate $F_S$ frequency. For maximum accuracy, the prescaler allows odd division by two (using the ODD bit in the SPI_I2SPR register).

Due to the SYSCLK low frequency (32 MHz Max.), the division results in a low accuracy factor leading to audio quality degradation. The accuracy factor decreases when the STM32L1xx I²S peripheral generates the master clock in master mode. Refer to the reference manual (RM0038: STM32L100xx, STM32L151xx, STM32L152xx and STM32L162xx advanced ARM®-based 32-bit MCUs) for the audio frequency accuracy depending on the targeted sampling frequency.

# 2 Implementation example

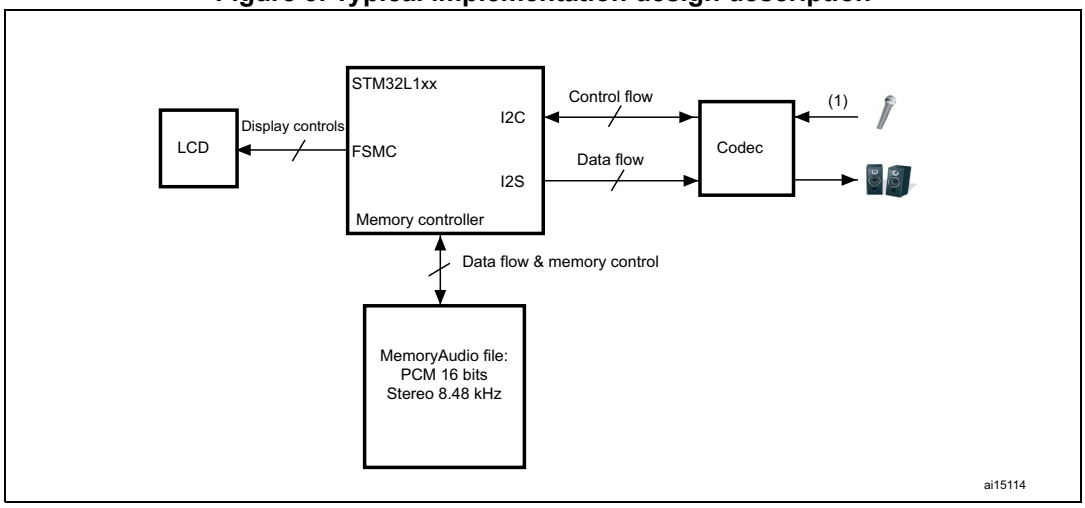
## 2.1 General overview

The example presented in this application note aims at providing typical hardware and software implementation basics for an audio application like portable audio players, sound synthesis systems, speech recorders, cell phones or interactive control boards.

Typically, the system embeds:

– a microcontroller (STM32L1xx device)
– an audio codec
– a speaker
– a memory support (where the audio file is stored).

**Figure 3. Typical implementation design description**




1. The audio input functionality (microphone) is not discussed in this application note.

The audio file format supported by the application is PCM, 16-bit data length, stereo/mono channels, 8 to 32 kHz audio frequency because the audio codec used in this application note needs to receive the master clock output.

## 2.2 Hardware description

### General requirements

The developed example is mainly based on the STM32L152D-EVAL evaluation board but the functional and structural description is similar for most applications and platforms.

The memory in which the audio file is stored is the NOR Flash memory implemented on the board. A different memory/source may be configured as the audio file support (like the SPI Flash memory).

Other board resources are used to interface the application:

- Audio codec: CS43L22 (see *Section 2.2.1*) implemented on the STM32L152D-EVAL and connected to the I2S2 port (and to relative passive components) and controlled by I2C1 interface.
- Stereo audio speaker and audio jack connected to the audio codec and implemented on the STM32L152D-EVAL.
- Joystick and key push-buttons: connected to the PG6, PG7, PG8, PG11 and PG13 pins on the board. These push-buttons are used to control the audio stream.
- LCD screen: implemented on the STM32L152D-EVAL evaluation board and controlled by the FSMC interface.

### 2.2.1 Audio codec

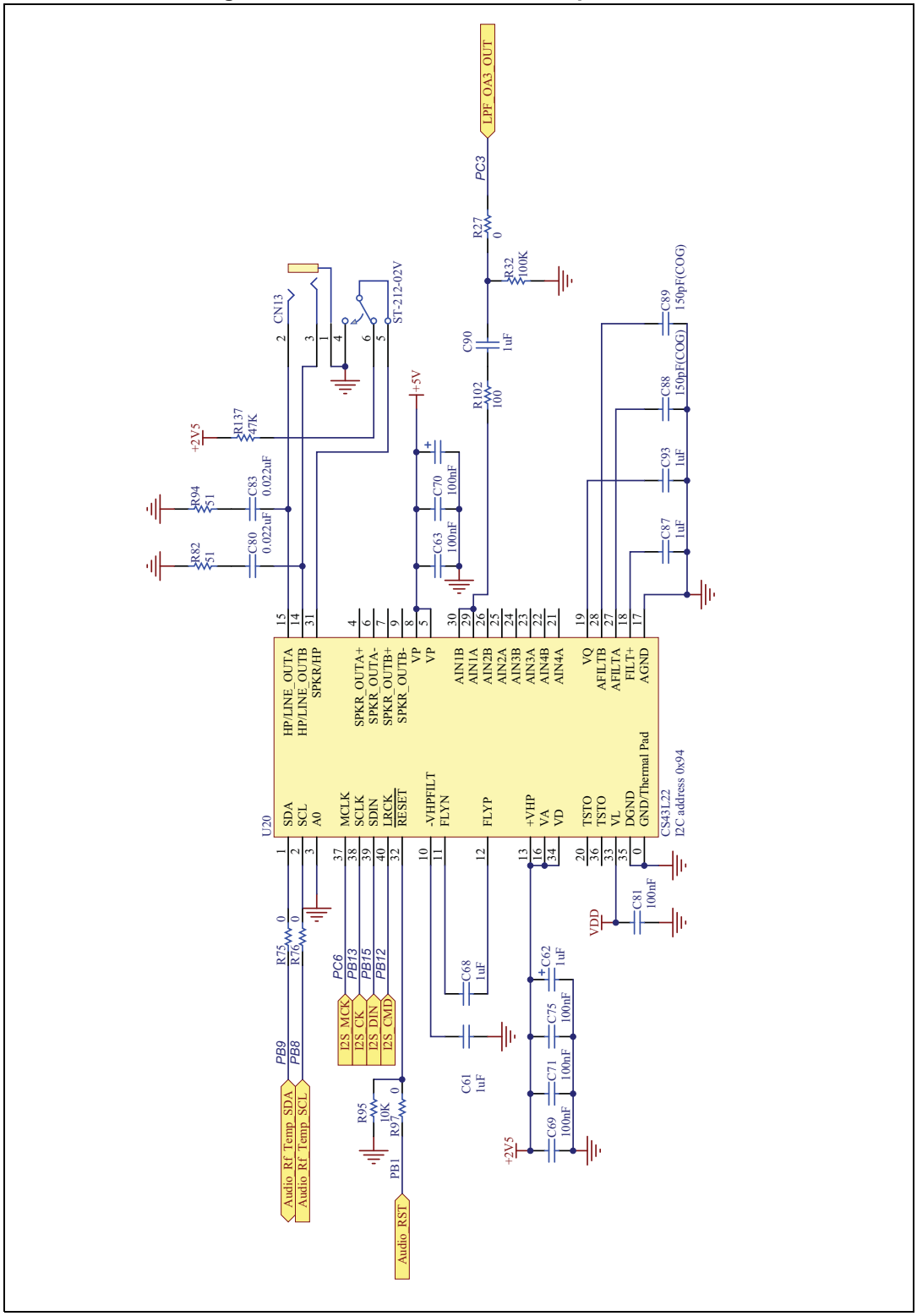
The audio codec implemented on the STM32L152D-EVAL is the CS43L22 from Cirrus Logic®. This codec allows digital (PCM raw data transmitted with $I^2S$ protocol) to analog conversion. The audio parameterization and the codec configuration are performed through an $I^2C$ interface. The codec has a wide set of configuration registers mainly used to:

- Program the audio output (speaker or headphone) and input (analog input or digital $I^2S$ data, etc.).
- Select the slave or master operating mode, set the reference clock for internal and sampling operations.
- Set the digital volume level, the mute status etc...

The codec can operate in Master or Slave mode. In this application example, the codec is used in Slave mode, the STM32L152D-EVAL $I^2S$ being Master.

*Figure 4* illustrates the hardware implementation schematic and how the codec is connected to the STM32L1xx and the board components.

**Figure 4. Audio codec hardware implementation**



The default codec configuration used in this application note is:

- $I^2S$ standard: $I^2S$ Phillips (can be changed to MSB or LSB standards)
- MCLK clock provided by the STM32L152ZD(T6) microcontroller has a ratio equal to 256 × $F_S$ (sampling frequency).
- The codec detects the headphone or the speaker automatically.

### Codec configuration steps

The function named `Codec_Init` is called to initialize these different steps.

- GPIO configuration to connect the I2S2, the I2C (clocks + data) to the audio codec.
- Reset the codec registers following its power down/power up sequence.
- Configure the $I^2C$ to initialize the audio codec control interface.
- Keep the codec in Power off mode during codec initialization:
  - Automatic detection between Headphone or speaker by writing the power control 2 register at address 0x04.
  - Slave mode and $I^2S$ standard protocol selection by writing the Interface Control 1 register at address 0x06.
  - Volume control configuring the Master Volume Control register at address 0x20 and 0x21.
  - Power on the codec writing the Power Control 1 codec register at address 0x02.
- Configure the $I^2S$ peripheral of the STM32L152.
- Send the audio data through the $I^2S$ interface and stop $I^2C$ communications.

In the `stm32l152d_eval_audio_codec` driver file, a single function performs the codec configuration:

```
Codec_Init(uint16_t OutputDevice, uint8_t Volume, uint32_t AudioFreq)
```

## 2.2.2 STM32L1xx and board configuration

The STM32L1xx peripherals used for this application are: I2S2 for audio communication, I2C1 for codec configuration and the memory interface (could be FSMC for NOR Flash memory or SPI1 for SPI_Flash memory, etc.).

- If the size of the audio file to be played is big, it should be previously loaded into the memory source (NOR Flash memory or SPI Flash) using an independent application (IAP, DFU, etc.). It may also be included as a table file.

For further details, please refer to the *STM32L152D-EVAL demonstration firmware User Manual* (UM1510) on www.st.com.

## 2.3 Firmware description

This application note is based on the audio part extraction of the STM32L152D-EVAL evaluation board firmware project. It is mainly based on:

- the STM32L1xx firmware library
- the `stm321152d_eval_audio_codec` driver firmware (offering the main functions required to control the codec and $I^2S$ environment for an audio application)
- a specific firmware to call the `stm321152d_eval_audio_codec` driver functions, as well as other functions required for control and display (`main.c`, `stm32l1xx_it.c files, waveplayer.c`).

The user may build any similar application using the same library and driver, and different interfacing firmware/hardware.

### 2.3.1 `Stm321152d_eval_audio_codec` driver firmware description

The user may interface the audio codec directly through the driver application layer. The driver functions are summarized in the following sections. *Table 1* presents the general driver file organization.

**Table 1. Driver library description**

| File | Description |
|---|---|
| `stm321152d_eval_audio_codec.h`, `stm321152d_eval_audio_codec.c`<br>waveplayer.c<br>waveplayer.h | – $I^2S$ and codec definitions, type definitions and function prototypes<br>– Basic functions (init, read, write, play, pause, stop, etc.). |

#### High-level functions

These are the functions that can simply be called by the final application to execute all needed configurations and perform high-end functionalities (like playing a wave sound, pausing playing, configuring all the hardware components, etc.).

These functions are presented in *Table 2*.

**Table 2. `Stm321152d_eval_audio_codec` driver high-level functions**

| Function name | Description |
|---|---|
| `EVAL_AUDIO_Init` | Initializes the entire application environment ($I^2S$, $I^2C$, codec, memory) |
| `EVAL_AUDIO_DeInit` | Deinitializes all the resources used by the codec |
| `EVAL_AUDIO_Play` | Causes the audio file to start playing |
| `EVAL_AUDIO_PauseResume` | Pauses the audio stream playing and saves the current position. It must be called for resume. |
| `EVAL_AUDIO_Stop` | Causes the audio file to stop playing and resets all local pointers. It powers down the Audio codec in addition. |
| `EVAL_AUDIO_VolumeCtl` | Increases/Decreases/Sets the digital volume |
| `EVAL_AUDIO_Mute` | Causes the codec to mute the released sound. |

Only the most relevant functions will be detailed in the following sections.

- **`EVAL_AUDIO_Init` function**

  This function implements all the needed initialization for the $I^2S$, the codec and the memory interfaces and peripherals.

**Table 3. `EVAL_AUDIO_Init` function**

| Function name | `EVAL_AUDIO_Init` |
|---|---|
| Prototype | `uint32_t EVAL_AUDIO_Init(uint16_t OutputDevice, uint8_t Volume, uint32_t AudioFreq)` |
| Behavior description | Initializes the $I^2S$ and the $I^2C$ peripherals, the codec, the memory and the audio file. |
| Input parameters | OutputDevice: used to set the output device, can be:<br>– OUTPUT_DEVICE_SPEAKER<br>– OUTPUT_DEVICE_HEADPHONE<br>– OUTPUT_DEVICE_BOTH<br>– OUTPUT_DEVICE_AUTO<br>Volume: Initial volume level<br>AudioFreq: Audio frequency to play the audio stream |
| Output parameter | None |
| Return value | – 0: if all initializations were successful<br>– 1: if initialization phase fails |
| Required preconditions | None |
| Called functions | Codec_Init, Audio_MAL_Init |

This function calls some subfunctions related to each component:

– `Codec_GPIO_Init()`: Configures the codec related IOs
– `Codec_Reset`: Resets the codec registers.
– `Codec_CtrlInterface`: Initializes the control interface of the Audio Codec.
– `Codec_VolumeCtrl`: Sets the volume.
– `Codec_AudioInterface_Init`: Configures the $I^2S$ peripheral.

- **`EVAL_AUDIO_Play` function**
  This function starts playing the audio file from a programmable position.

**Table 4. `EVAL_AUDIO_Play` function**

| Function name | `EVAL_AUDIO_Play` |
|---|---|
| Prototype | `uint32_t EVAL_AUDIO_Play(uint16_t* pBuffer, uint32_t Size)` |
| Behavior description | Causes the audio file to start playing from a data buffer for a determined size |
| Input parameter | pBuffer: pointer to the buffer<br>Size: Number of audio data bytes |
| Output parameter | None |
| Required preconditions | None |
| Called functions | `Audio_MAL_Play, Codec_Play` |

- **`EVAL_AUDIO_VolumeCtl` function**
  This function controls the digital volume level in accordance with the input parameter.

**Table 5. `EVAL_AUDIO_VolumeCtl` function**

| Function name | `EVAL_AUDIO_VolumeCtl` |
|---|---|
| Prototype | `uint32_t EVAL_AUDIO_VolumeCtl(uint8_t Volume)` |
| Behavior description | Sets a volume level defined by the volume variable. |
| Input parameter | – volume: From 0 to 100 (0: min level (0%), 100: Max level (100%)) |
| Output parameter | None |
| Required preconditions | None |
| Called functions | `Codec_VolumeCtl` |

## 2.4 Demo firmware description

This application firmware is extracted from the STM32L152D-EVAL evaluation demonstration firmware. For further details, please refer to the *STM32L152D-EVAL demonstration firmware User Manual* (UM1510), *Section 2.4.4 Wave player submenu* and *Section 4.1 Programming the media files*.

The media directory in the STM32L152D-EVAL demonstration firmware must be loaded into the SD card. This directory contains BMP files (*.bmp) for the LCD screen and audio files (*.wav) to be played.

# 3 Revision history

Table 6. Document revision history

| Date | Revision | Changes |
|---|---|---|
| 01-Aug-2013 | 1 | Initial release. |
| 27-May-2014 | 2 | Updated introduction and disclaimer. |