Audio Output Function Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Characteristics

The audio output library is a low-level audio library for outputting PCM audio data. It can output 16-bit linear PCM, 2-channel stereo, or 1-channel monaural audio.

Audio output of the PCM output from the audio synthesis library or the audio decoder library executed on PlayStation®Vita can be done using this library.

The library has three ports, MAIN, BGM and VOICE. Which port is used depends on the function.

MAIN Port

This port is intended for the audio output of sound effects and speech. It can open up to 8 channels of signed 16-bit stereo or monaural audio and mix them for playback. The sampling frequency is fixed at 48000 Hz.

BGM Port

This is the BGM playback port. It can play back 1 channel of signed 16-bit stereo or monaural audio. Sampling rate conversion function is provided for the BGM port output. Output to the BGM port may be switched to different audio for system reasons. For example, the audio of the music player in the system may be switched to. The audio of the BGM port is output mixed with the audio of the MAIN port.

The BGM port has a dynamic normalizer (ALC: Automatic Level Control) effect. You can enable or disable the ALC function by calling `sceAudioOutSetAlcMode()`.

VOICE Port

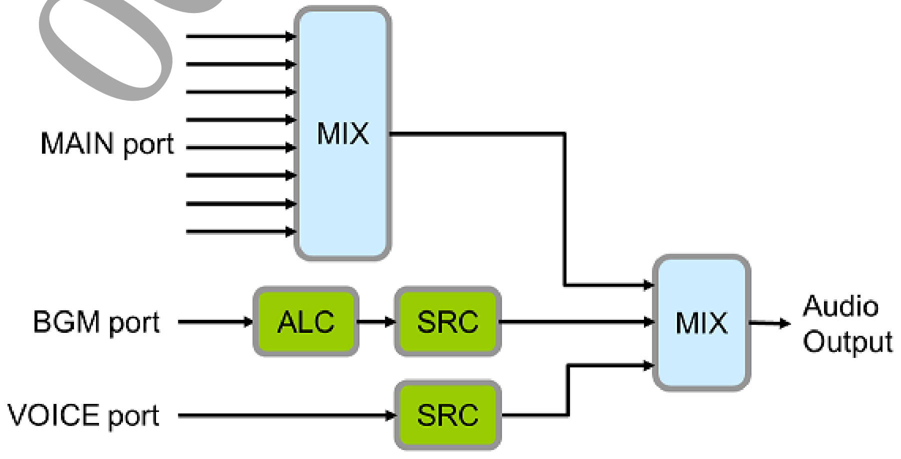
This is the voice chat audio playback port. It functions almost the same as the BGM port. But the condition for switching the audio output to this port is unlike the BGM port. The audio for the system voice chat function is output. This audio is mutually exclusive with that of other voice chat applications.

Sampling Rate Conversion (SRC)

The BGM port and VOICE port have a sampling rate conversion function. It can output audio sampled at 48000, 44100, 32000, 24000, 22050, 16000, 12000, 11025, and 8000 Hz.

The overall configuration of the audio output is as shown below.

Figure 1 Connection Configuration



Files

The files required to use the audio output library are as follows.

File	Description
audioout.h	Header file
libSceAudio_stub.a	Stub library file

Sample Programs

The sample program for the audio output library is as follows.

sample_code/audio_video/api_audioout/stream/

This sample program shows the basic way in which the audio output library can be used.

Reference Materials

For details about the various functions, refer to the "Audio Output Function Reference" document.

2 Using the Library

Basic Procedure

(1) Initialize library

First, call `sceAudioOutOpenPort()` and perform initialization. To output audio, specify the sample length to be output at one processing, the sampling rate, etc., along with declaring the use of the audio output function with this function. (For details, refer to the "Audio Output Function Reference" document.

Example:

```
int port_id;

/* Output 256 samples at one time */
port_id = sceAudioOutOpenPort(
  SCE_AUDIO_OUT_PORT_TYPE_MAIN,
  256, 48000,
  SCE_AUDIO_OUT_PARAM_FORMAT_S16_STEREO);
if (port_id != SCE_OK){
  /* Setting failed */
}
```

(2) Audio output

Audio output processing is done with `sceAudioOutOutput()`. The PCM audio data stored to the specified buffer is registered to the audio driver. The registered sound data starts playing back after the sound data already registered to the audio driver has been played back. Moreover, the thread that called this function is in the WAIT state until playback starts.

Playback does not necessarily start immediately following registration to the audio driver. Also, at the time the processing returns from the function, playback of the registered sound data starts, but this does not mean that playback has been completed, and neither does this mean that that data has been copied somewhere. Therefore, if the contents of the buffer are rewritten before the actual end of playback, the data is not correctly played back. Set the buffer for sound output to double buffer or greater, and be careful not to rewrite the buffer contents immediately after registration.

Example:

```
/* Set pcmBuf to double buffer */
int iSide = 0;

/* Output 16 bit PCM stereo data in pcmBuf[iSide] */
sceAudioOutOutput(
  port_id,
  pcmBuf[iSide]
);
iSide ^= 1;
```

(3) End processing

End processing is done with `sceAudioOutReleasePort()`.

If `sceAudioOutReleasePort()` is called in before playback completion of sound data registered to the audio driver, end processing will fail and an error will return. Moreover, when the PCM data has been placed in a memory area allocated dynamically (area allocated with the `malloc()` function, automatic variable allocated within the function, etc.), if the memory area is freed before output of the data has ended, noise that differs according to the conditions during execution may be output.

In order to prevent this, check whether output of the last data has indeed ended before freeing the memory. Caution is also required for the handling of threads. A dedicated thread is often used for PCM sound output. In such a case, in order to prevent forced end of the sound output thread while data is still being output, have the processing done with synchronization between the threads.

An effective method to wait until the sound output has ended is to hand over the NULL pointer to the data argument of the `sceAudioOutOutput()` function.

Example:
```
/*  Output processing */
for( ; ; ){

  sceAudioOutOutput(port_id, pcmBuf[iSide]);
  iSide ^= 1;
}
/* Last data was passed */

/* Wait for the output of the data not yet output to complete */
sceAudioOutOutput(port_id, NULL);

/* Release port */
sceAudioOutReleasePort(port_id);
```

List of Functions

Initialization and Termination Functions

Function	Description
`sceAudioOutOpenPort()`	Initializes the channels of the port of the specified type
`sceAudioOutReleasePort()`	Closed the port and releases the channels

Output Function

Function	Description
`sceAudioOutOutput()`	Outputs the specified waveform data from the user

Volume Setting Function

Function	Description
`sceAudioOutSetVolume()`	Sets the volume for each channel

Parameter Setting and Setting Acquisition Functions

Function	Description
`sceAudioOutSetConfig()`	Can set again a number of parameters
`sceAudioOutGetConfig()`	Gets the current value of each parameter

Remaining Number of Samples Acquisition Function

Function	Description
`sceAudioOutGetRestSample()`	Gets the number of samples that have not been played back among the registered samples

3 Cautions

Granularity and Processing Load

The *`len`* argument specified with `sceAudioOutOpenPort()` is the number of samples processed each time the `sceAudioOutOutput()` function is called. This number of samples is called "granularity."

The granularity setting value can be specified in multiples of 64, but caution is required when setting this value. If a small value (for example, the minimum value of 64) is specified, the time until the processing returns from the `sceAudioOutOutput()` function is short, and the time from when the function is called until output can thus be shortened. However, it should be noted that this also means that the number of times the function is called per given interval of time is higher, which places a greater average processing load on the CPU. However, simply increasing the granularity of a given output channel does not necessarily reduce the load on the CPU. If the timing matches that of other outputs, the load can be further reduced. For example, rather than combining 64*4=256 granularity and 64*5=320 granularity, setting the granularity of both outputs to 64*4=256 results in a matching output timing that reduces the load on the CPU.

Calling Multiple Output Functions within a Thread

For the purpose to mix multiple audio within a thread, be sure not to use multiple channels of the MAIN port. Similarly, do not use the output of different port types together within a thread (such as MAIN port and BGM port).

Output function `sceAudioOutOutput()` is a blocking type function, so when this function is called more than one time within a thread, the playback sound is interrupted. Therefore, multiple audio cannot be mixed within a thread. To generate mixed audio using a single output function, generate the sound by preparing the mixed audio in advance.