SAS Overview

PlayStation®Vita Programmer Tool Runtime Library Release 3.50

Table of Contents

1 Library Overview

Functional Overview and Features

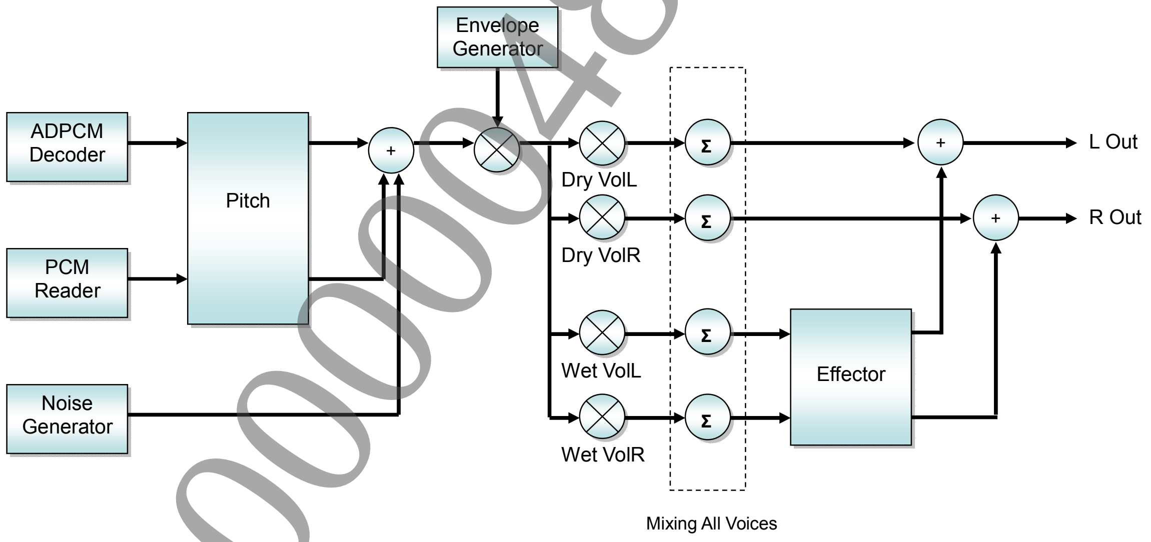
SAS is a library for performing synthesizer processing with software. SAS enables waveform data such as ADPCM (.vag) to be used as phoneme data for performing functions such as waveform data sound generation, pitch change, envelope processing and digital effects on voice data.

Features

- The output data format is 48kHz 16-bit 2-channel (with effects for stereo mode) or 4-channel output (without effects for multichannel mode)
- The granularity (number of PCM data samples that are generated each time internal processing is executed) can be selected from 64 to 2048 in multiples of 32.
- Supported waveform formats are ADPCM (.vag) and PCM (16-bit monaural)
- Built-in noise sound source
- The pitch can be controlled from the -12 octave to the +2 octave
- Built-in digital effects functions
- Built-in ADSR mode envelope
- Built-in pause function for a voice that is being vocalized

SAS Internal Block Diagram

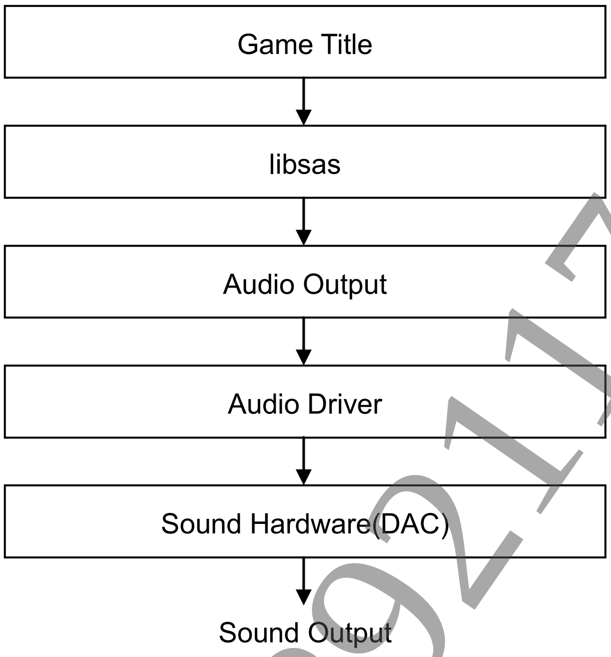
The following figure shows an internal block diagram of SAS.



Processing Flow Up to Sound Output

The following figure shows the processing flow up to sound output of SAS.

Figure 1 Processing Flow Up to Sound Output



Related Files

The following files are required to use SAS.

Filename	Description
libSceSas_stub.a	Library
libSceSas_stub_weak.a	Weak import library
sas.h	Header file

Sample Programs

The following sample programs use SAS.

sample_code/audio_video/api_libsas/basic

This is a simple sample that plays back a one-shot waveform.

2 Waveform Data Format

For the waveform data that is the sound source of each voice, SAS uses either PCM data, or audio data that was compressed using ADPCM (.vag). Looping of the playback is supported so that it can easily be used as the synthesizer sound source.

ADPCM Data Block Format

Waveform data consists of blocks of 16 bytes each. A block consists of a 2-byte header in which the decoding coefficient and loop information are recorded and 14 bytes of sound data, comprising 28 samples worth of audio data.

Figure 2 ADPCM Data Format

Decoding coefficient and loop information (2 bytes)	Sound data (14 bytes)

ADPCM Data Loop Information

The low-order 4 bits of the second leading byte of each block of the ADPCM data are the loop information. By setting appropriate data for loop information, vocalization data loops can be implemented in terms of individual blocks.

How this value is used differs according to the playback mode that can be set for each voice. The two playback modes are "loop play" and "one-shot play." Interpretations of each mode are explained below.

The size of phoneme sample data is specified when the phoneme sample is set. When decoding this amount of data ends, vocalization ends and the relevant voice is muted regardless of the loop information value or whether loop playback is enabled or disabled.

When loop play is enabled

Loop information is interpreted as shown in the following table. Values not in the table are ignored.

Loop Information Value	Meaning
3	Indicates the ending block of the loop. After this block is decoded, decoding returns to the starting block, which was saved internally. If the ending block is reached without finding the starting block, decoding returns to the phoneme sample starting position that was set when the phoneme sample was specified.
6	Indicates the starting block of the loop. When the ending block is reached, this becomes the block that is decoded next. If multiple starting blocks existed, the one that was traversed last is valid.
7	Indicates the playback ending position. When this block is reached, vocalization of the relevant voice is stopped and the voice is muted. As a result, this block is not decoded.

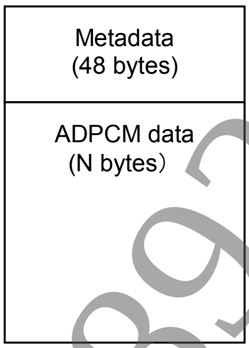
When loop play is disabled (one-shot play)

Loop information is interpreted as shown in the following table. Values not in the table are ignored.

Loop Information Value	Meaning
7	Indicates the playback ending position. When this block is reached, vocalization of the relevant voice is stopped and the voice is muted. As a result, this block is not decoded.

Creating ADPCM Data

The tool such as VAG Converter 2 is provided for creating ADPCM data from PCM data that can be used by SAS. For details, refer to "VAG Converter 2 Tools User's Guide" document. The VAG Converter 2 also outputs a VAG file that contains a header in which metadata such as the original data is stored, followed by the ADPCM data.



The format of the metadata is as follows.

```
typedef struct SceVagHeader {
        SceUInt8 VAGp[4];          // 'VAGp' fixed
        SceUInt32 version;         // 0x00020001 fixed
        SceUInt8 _reserved1[4];
        SceUInt32 dataSize;        // Waveform size [byte]
        SceUInt32 fs;              // frequency [Hz]
        SceUInt8 _reserved2[10];
        SceUInt8 ch;               // Number of Channels (0 to 1=1[ch], 2=2[ch])
        SceUInt8 _reserved3;
        SceChar name[16];          // name (NULL termination)
} SceVagHeader;
```

*) All members of `SceVagHeader` are big-endian.

PCM Data Format

The PCM format used by SAS is a headerless, general 16-bit sampling, little-endian, monaural format. No information related to looping of the playback position is contained in the PCM data. An application that uses SAS must manage this information separately from the PCM data and must specify it when calling functions that set the phoneme sample.

3 Voice Processing

The processing that is performed for each individual vocalized voice (voice processing) consists of tasks for decoding waveform data, varying volume and pitch, assigning temporal variations, and generating actual vocalizations. Processing for an externally assigned vocalization starting request (key on) is also performed for each voice.

Sound Source Generation Processing

Waveform data is obtained according to the method specified as the sound source. PCM, ADPCM or noise generator can be selected for the sound source type for each voice. Although the sound source type normally can be changed, a click tone may be issued when switching occurs because switching is performed without specifically performing a fade in even when it is performed for a voice that is being vocalized.

When the sound source is the ADPCM, waveform data (PCM data) is obtained by decoding phoneme sample data, which is explained in the "Waveform Data Format" chapter. When the sound source is PCM, the PCM data that was input is used directly. When using these sound sources, the position of the phoneme sample data that should be decoded is set by using the voice setting function before sound is produced.

When the sound source is the noise generator, a pseudo-random number is generated by a calculation to obtain noise PCM data. Noise tones having different properties can be generated by externally assigning parameters used for the pseudo-random numbers.

Pitch Transformation

When the sound source is PCM or the ADPCM, the pitch can be varied for the decoded vocalization data in the maximum range from a 2 ($\log_2 2^{14}/2^{12}$) octave increase to a 12 (-$\log_2 1/2^{12}$) octave decrease.

The amount of pitch variation is specified by the pitch value P. The range of values that P can have is as follows.

$$1 \le P \le 16384 = 2^{14}$$

If f_0 denotes the original pitch of the phoneme sample and f_s (Hz) denotes its sampling rate, then the final pitch f of the voice that is vocalized is given as follows.

$$f = \frac{48000}{f_s}\frac{P}{2^{12}}f_0$$

Note that you can reduce the apparent data size by raising the pitch (making f_0 smaller) when preparing the phoneme sample data and then making P smaller during playback. However, since this will eliminate the high-frequency components according to sampling theory, sound quality will deteriorate.

Envelope Processing

Envelope processing simulates the reproduction of the key on to key off of existing musical instruments and the subsequent temporal variation of the vocalization volume. This temporally varying volume is also known as the envelope value. SAS divides the variation of the envelope value into the following four stages and enables various parameters to be set corresponding to each state.

(1) Attack
Stage from immediately after key on is performed until the envelope value reaches its peak

(2) Decay
Stage after Attack until the envelope value falls to a specified level (Sustain Level)

(3) Sustain
Stage after Decay until key off is performed

(4) Release
Stage after key off is performed until the envelope value becomes 0

For details about these parameters, see the "4 Envelope Processing" chapter.

Key On/Key Off

Key on (vocalization start) and key off (vocalization end) can be specified for each voice.

When key on is performed, processing concerning phoneme samples decoding, pitch, envelope, and volume starts.

When key off is performed, the envelope variation state of the relevant voice switches to release state and the vocalization volume varies according to the release rate (vocalization does not end immediately when key off is performed). Vocalization ends when the volume becomes zero in the release state.

Note that even before key off is performed, if the phoneme sample data ends (when there is loop information that means it ends or when the playing of the area specified as the size of the phoneme sample data ends), vocalization of the relevant voice ends there and the vocalization end flag of the relevant voice is set.

Volume

The volume of the left and right normal output (DryL and DryR) and left and right effects input (WetL and WetR) can be set independently for each voice. This enables a phoneme sample that was originally monaural to be made to have an arbitrary pan and be output as a stereo vocalization.

The specified value is directly used for the volume value. A temporally varying volume value cannot be assigned as with the envelope value.

Mixing

The outputs of each voice are mixed in the four channels DryL, DryR, WetL, and WetR.

Among the mixed results, the contents of the two channels WetL and WetR are entered in the digital effector, and after special effects are added, they are combined with DryL and DryR (described later).

4 Envelope Processing

Envelope processing, as described earlier, reproduces the temporal transitions of the vocalization volumes of numerous musical instruments. This chapter explains details of envelope processing.

State Transitions

As described earlier, SAS divides the envelope variation into four time bands.

The attack state occurs immediately after key on is performed. The envelope value varies in the attack state according to the variation format (attack shape) and variation rate (attack rate).

The transition to the decay state occurs when the envelope value reaches its maximum value. Similarly, the envelope value varies in the decay state according to the variation format (decay shape) and variation rate (decay rate).

Subsequently, the transition to the sustain state occurs when the envelope value reaches the sustain level, and the transition to the release state occurs in the state when key off was performed. The envelope value varies in each state according to the sustain shape and sustain rate and the release shape and release rate, respectively.

Parameters Related to Variation Rate

The size of the envelope value (temporal varying volume) variation per unit time can be assigned as a parameter.

Parameter	Meaning
Attack Rate (AR)	Variation rate of the envelope value in the attack state
Decay Rate (DR)	Variation rate of the envelope value in the decay state
Sustain Rate (SR)	Variation rate of the envelope value in the sustain state
Release Rate (RR)	Variation rate of the envelope value in the release state

Parameter Related to Variation Position

The value that is the boundary where the state switches from decay state to sustain state can be set externally.

Parameter	Meaning
Sustain Level (SL)	Envelope value that is the boundary where the state transitions from decay state to sustain state

Parameters Related to Variation Shape

In SAS, the following six variation shapes (modes) can be used.

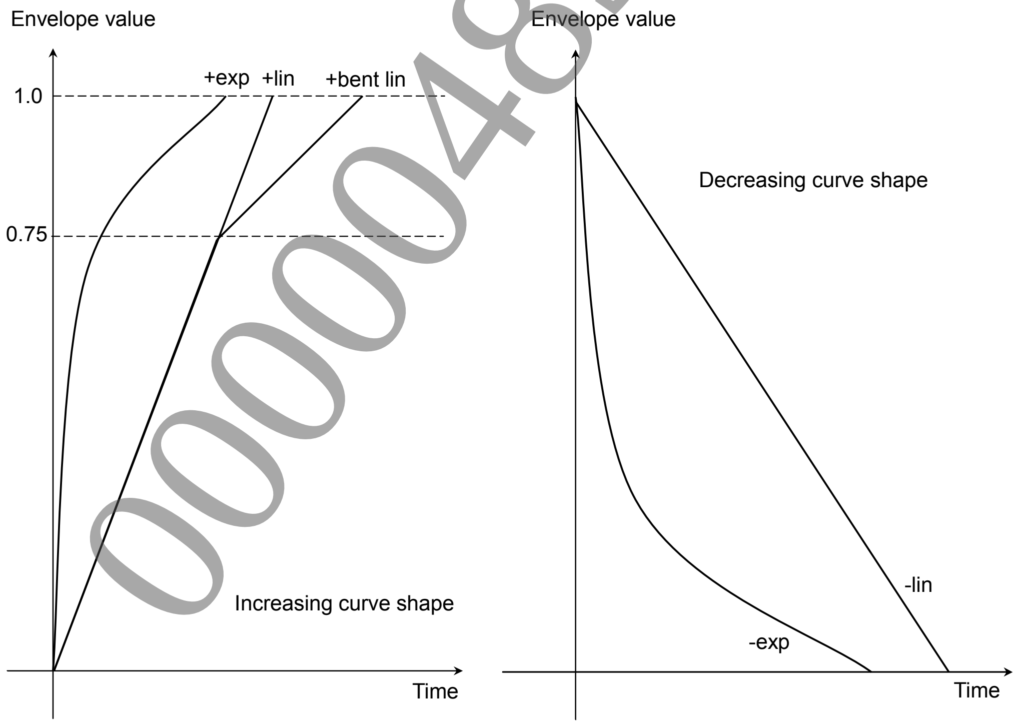
Variation Shape (Mode)
Linear increase (+lin)
Linear increase according to a broken line (+bent lin)
Exponential increase (+exp)
Linear decrease (-lin)
Exponential decrease (-exp)
Direct specification (direct)

The variation shape mode described here as a "Linear increase according to a broken line" indicates an increase mechanism in which the rate of increase gets smaller when 75% of the maximum envelope value is reached. Also, when a "Direct specification" is used, the envelope value can be directly changed to the value that was assigned as the variation rate.

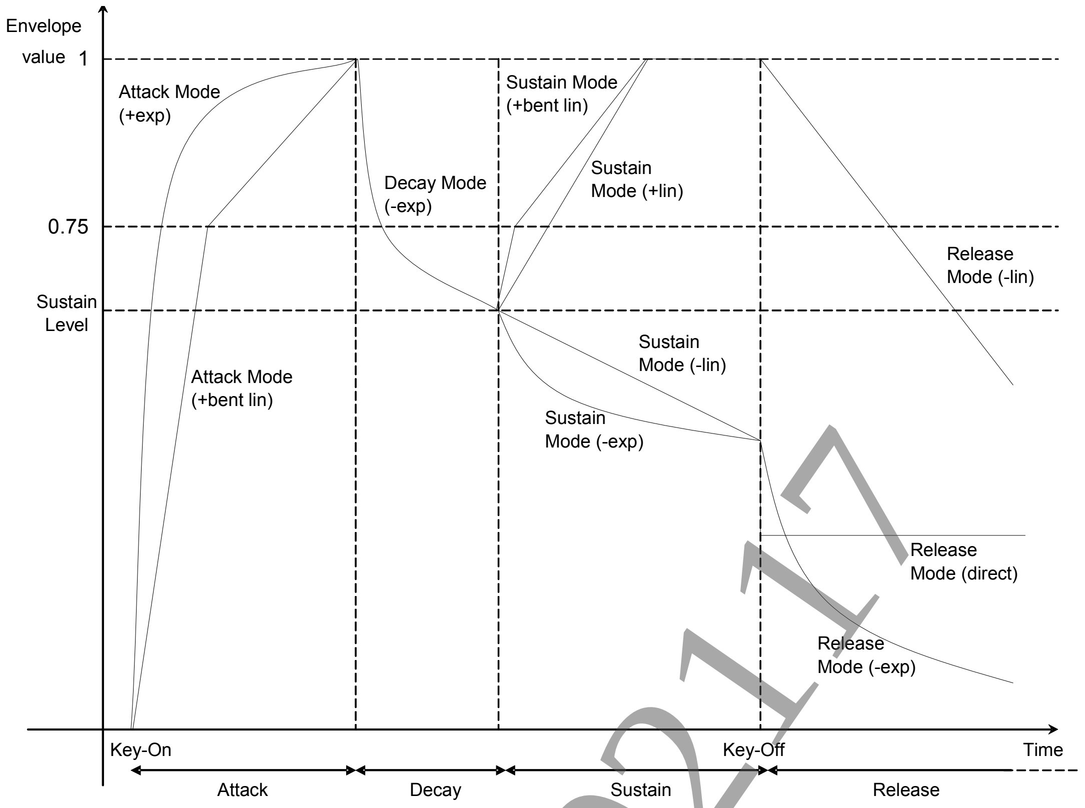
Although one variation shape (mode) can be selected for each of the four states of envelope processing, the shapes (modes) that can be selected are limited as follows according to the state that is in effect.

Parameter	+lin	+bent lin	+exp	-lin	-exp	direct
Attack mode (AM)	○	○	○	---	---	---
Decay mode (DM)	---	---	---	○	○	○
Sustain mode (SM)	○	○	○	○	○	○
Release mode (RM)	---	---	---	○	○	○

Each shape (mode) is illustrated in the following figure. The *y*-axis value in this figure indicates a value relative to the maximum envelope value.



Also, the following figure shows examples of the relationships between each parameter and the state transitions of the four states that represent the envelope. The *y*-axis value in this figure indicates a relative value to the maximum envelope value.



Note that in the Sustain and Release states, if the envelope value reaches 0, vocalization of the relevant voice ends there, and 1 is set for the vocalization end flag.

Envelope Specification Methods

SAS enables two methods to be used to specify envelope parameters.

Detailed method

This is a method in which four parameters related to the variation rate (AR, DR, SR, RR), one parameter related to the variation position (SL), and four parameters related to the variation shape (AM, DM, SM, RM) are set by using the respective dedicated functions.

Although all functions that SAS has can be set, the code becomes somewhat complex.

The larger the value specified to AR, DR, SR, and RR, the larger the variation rate of the corresponding state.

The larger the value of SL, the higher the sustain level.

Simple method

This is a method in which all parameters are specified by using the two 16-bit values ADSR1 and ADSR2. The two values become the following bit field values. Since the numbers of bits allotted to each parameter are less than the numbers that can represent all functions or the resolution that SAS has, only rough specifications can be made. However, the parameters can be set easily.

Figure 3 ADSR1 Bit Assignments

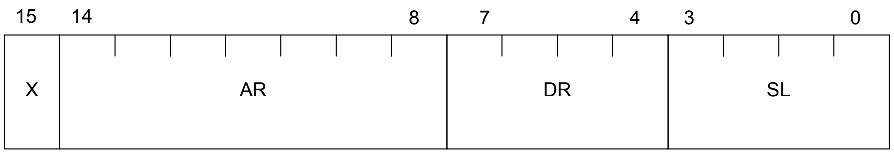
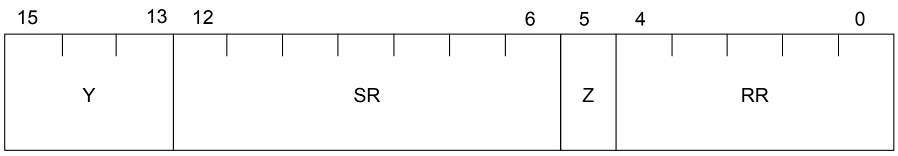


Figure 4 ADSR2 Bit Assignments



This is different from the Detailed method in that the smaller the value specified to AR, DR, SR, and RR, the larger the variation rate of the corresponding state.

The larger the value of SL, the higher the sustain level.

SL is the parameter for specifying the sustain level. The larger the value, the higher the sustain level.

X, Y, and Z are parameters for specifying the curve shape.

X indicates the attack shape, where 0 is linear increase and 1 is linear increase according to a broken line.

Y indicates the sustain shape, where 000b is linear increase, 010b is linear decrease, 100b is linear increase according to a broken line, and 110b is exponential decrease. If any other value is specified, an error occurs.

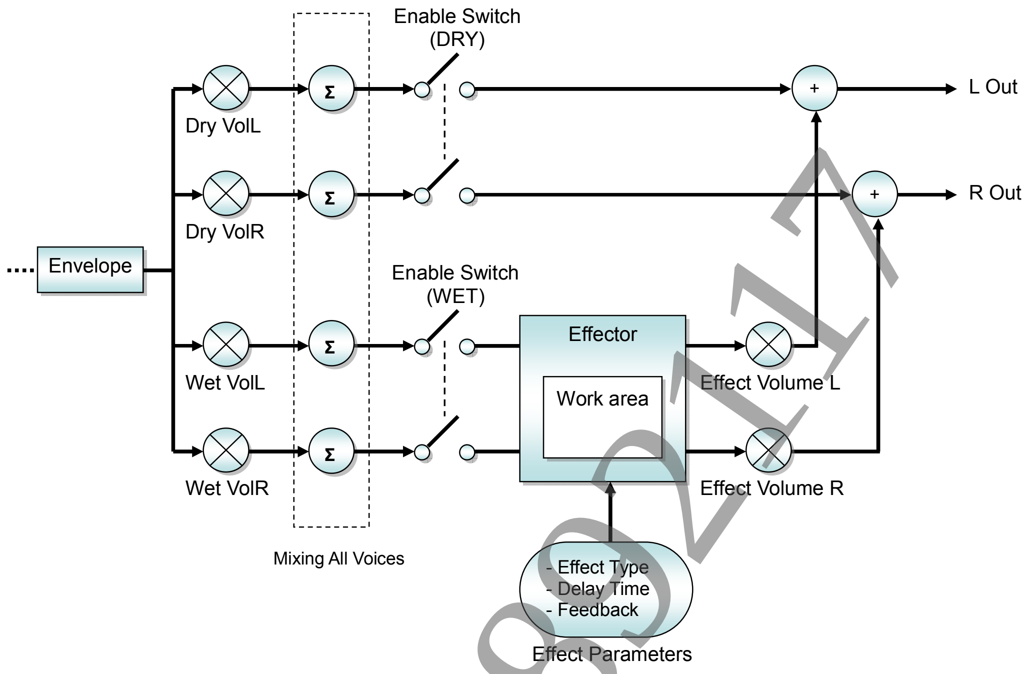
Z indicates the release shape, where 0 is linear decrease and 1 is exponential decrease.

Note that when the simple method is used for specifying the envelope variation, the decay shape is always an exponential decrease.

5 Digital Effects Processing

After processing is performed for each voice, various acoustic effects such as reverb, echo, and delay can be added by applying digital effects to the results of mixing the outputs of the various voices.

The following figure shows a block diagram of the blocks related to the effector.



To output the output of each voice via the digital effector, the volume of the relevant voice to be sent to the effector (Wet Vol L and Wet Vol R) must be set appropriately and the effector enable switch must be set to On. Also, by changing the balance with the volume for the output components that do not pass through the effector (Dry Vol L and Dry Vol R), the conditions for applying the effect can be varied for each voice.

The contents of the effects to be applied can be changed by assigning parameters corresponding to the effect type, delay time, and feedback coefficient for the effector. The volume for the effect output (Effect Volume L and Effect Volume R) can also be specified.

Digital Effect Types

One of the following 10 effect types can be selected.

If the effect type is OFF, the sound that is input is passed directly through without applying any effects.

A reverberation effect is added for all the other effects.

Effect Type
OFF
Room Reverb
Studio Reverb A
Studio Reverb B
Studio Reverb C
Hall Reverb
Space Echo
Echo
Delay
Pipe Echo

Effect Parameter

If Echo or Delay is selected for the effect type, the Delay time and Feedback can be selected. These settings are ignored for other effects.

If a larger Delay Time value is set, a longer delay interval for the overlaid sound will occur.

If a larger Feedback value is set, a greater feedback level and stronger effect will occur.

Parameter Name	Setting Range
Delay Time	0 to 127
Feedback Level	0 to 127

Input Switch and Effect Volume

Enable switch (DRY, WET)

These switches can turn on or off the sound to which the effect is not applied (Dry side) and the sound to which the effect is applied (Wet side). Both switches have the same value in the left and right channels.

Effect Volume (L, R)

The Effect Volume (L, R) can set the output level of the effect processing result.

Settings When Initialization is Performed

The following shows the effect settings when initialization is performed

Parameter Name	Initial value
Effect Type	OFF
Enable switch (DRY)	ON
Enable switch (WET)	OFF
Effect Volume L	0
Effect Volume R	0

6 Output Mode

The number of SAS output channels can be changed by switching the output mode. Output mode switching is useful when adding separate effects to individual sound sources. The available output modes are stereo mode and multichannel mode. The following figure shows the differences between the output modes.

Figure 5 Stereo mode

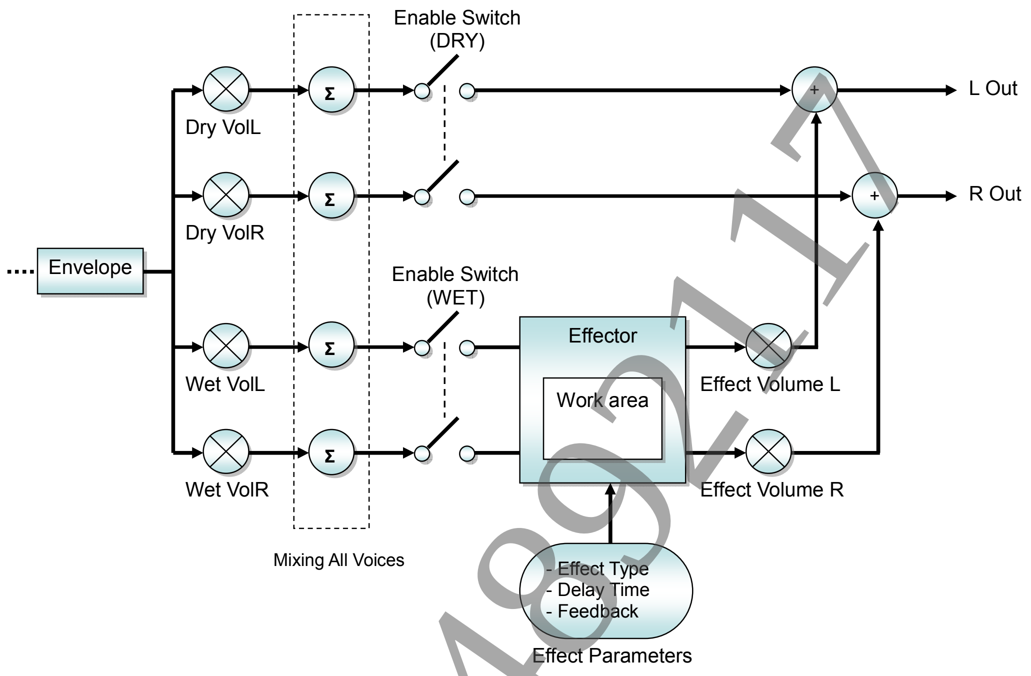
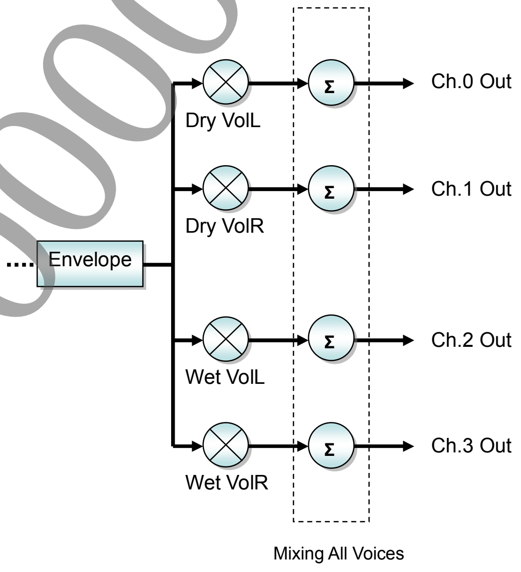


Figure 6 Multichannel mode



Stereo Mode

Stereo mode is the output format that had previously been provided by SAS. Its audio data flow is the same as that shown in the block diagram in the "Digital Effects Processing" chapter in this document. An output voice is a mixture of a dry voice and a wet voice which is a created effect based on the dry voice. The number of output channels is two, consisting of a left channel and a right channel.

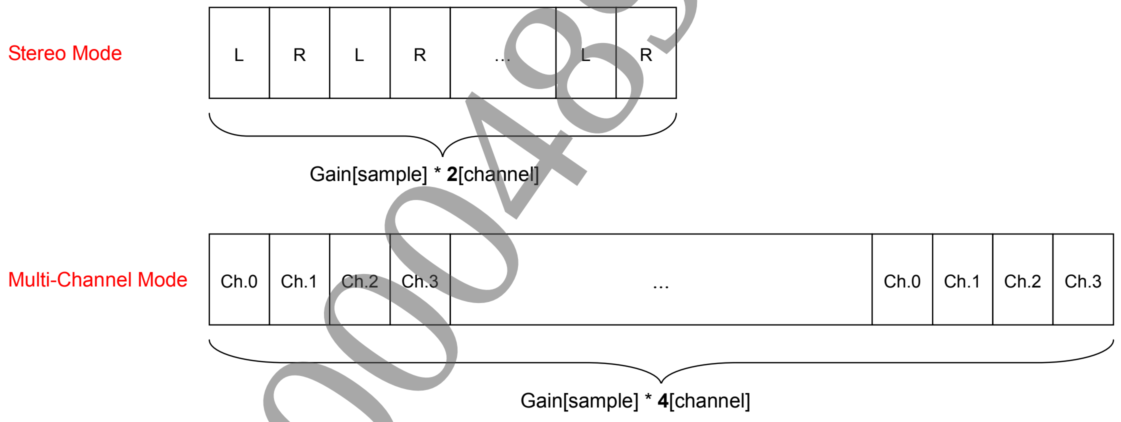
Multichannel Mode

In multichannel mode, the audio data that appears before the Enable Switches in stereo mode is output directly. In multichannel mode, effects are not added to any output voice, and parameters related to effects cannot be changed.

The number of output channels is four. The assignments to the various channels of the output voice after envelope processing are determined by the four volumes indicated by Dry Vol L, Dry Vol R, Wet Vol L, and Wet Vol R. Although the names Dry L, Dry R, Wet L, and Wet R are appended to each channel for convenience to make it easier to understand the correspondence with stereo mode, all four channels are equivalent and the role of each channel cannot be restricted.

An example of a multichannel mode application is the 3D positional audio playback of a moving sound source. When the moving sound source is reproduced, the listener has a significant feeling of realism in addition to a sense of orientation because the Doppler effect of the moving sound source is simulated. SAS can implement the Doppler effect through processing that changes the pitch.

SAS enables the two output modes to be dynamically switched. However, note that if the number of channels changes, the buffer size required for the output buffer also changes. In addition, the ordering of the PCM data that is output to the buffer changes as follows.



Although stereo mode output PCM data can be directly passed to an audio output function, multichannel mode output PCM data cannot. Multichannel mode does not take into consideration that sound is directly produced by the output voice, but assumes that the sound is produced after adding some kind of effect.

7 Overview of Usage Procedure

Procedure for Using SAS

The following procedure shows how to use SAS.

(1) Use `sceSasGetNeededMemorySize()` to obtain the memory size required for initialization.

(2) Use `malloc()` or other memory allocation function to allocate the memory required for initialization.

(3) Pass the buffer of (2) above and the size of (1) above to `sceSasInit()` and perform initialization.

(4) `Use sceSasSetVoice()` to set the basic parameters for the voice for which sound is to be produced

(5) Use a function such as `sceSasSetKeyOn()` to perform various types of sound processing (such as key on processing).

(6) Use `sceSasCore()` to generate PCM sound based on key on and parameter change information.

(7) Use `sceAudioOutOutput()`to output the generated PCM sound

(8) Use `sceSasExit()` to perform termination processing.

Initialization and Termination Processing

To use SAS, `sceSasGetNeededMemorySize()` must be used to obtain the memory size required for initialization, `malloc()` or another function must be used to allocate the required memory, and `sceSasInit()` must be used to perform initialization processing.

After using libsas, use `sceSasExit()` to perform termination processing when the application ends.

Example:

```
SceSasResult result;
size_t bufferSize;
void *buffer;

/* Get memory size required for initialization */
result = sceSasGetNeededMemorySize("", &bufferSize);
if (SCE_SAS_FAILED(result) {
        /* Failed to get memory size required for initialization */
}

/* Allocate memory for initialization */
buffer = malloc(bufferSize);
if (buffer == NULL) {
        /* Memory allocation Failed */
}

/* Initialization processing */
result  = sceSasInit("", buffer, bufferSize);
if (result != SCE_OK){
        /* Initialization failed */
}

        :

/* Describe processing for using libsas functions here */

        :
```

```
/* Termination processing */
result = sceSasExit(&buffer, &bufferSize);
if (result != SCE_OK){
        /* Termination processing failed */
}

/* Free memory used for initialization */
free(buffer);
```

Method of Outputting PCM Sound That was Generated by `sceSasCore()`

With SAS, PCM sound is generated by the sceSasCore() function. The generated PCM sound is output by using functions such as the audio output functions. To output PCM sound, first open the port and set the output volume.

Example:

```
int portId, result, aVolume[2];

/* MAIN, 48kHz, 256sample length, 16bit, Open stereo audio port */
portId = sceAudioOutOpenPort(
        SCE_AUDIO_OUT_PORT_TYPE_MAIN,
        256,
        48000,
        SCE_AUDIO_OUT_PARAM_FORMAT_S16_STEREO);
if (portId != SCE_OK){
        /* Open failed */
}
/* Output at 0dB */
aVolume[0] = SCE_AUDIO_VOLUME_0DB;
aVolume[1] = SCE_AUDIO_VOLUME_0DB;
result = sceAudioOutSetVolume(portId, SCE_AUDIO_VOLUME_FLAG_L_CH |
SCE_AUDIO_VOLUME_FLAG_R_CH, aVolume);
if (result != SCE_OK){
        /* Setting failed */
}
```

An audio output function such as `sceAudioOutOutput()` is used to output PCM-format sound. `sceAudioOutOutput()` registers the PCM-format sound that is stored in the specified buffer to the audio driver. Playback of the registered sound data is started after sound data that had already been registered in the audio driver is played.

Audio is not necessarily played back immediately because it was registered in the audio driver. Also, the audio driver does not create a copy of the sound data. As a result, if the buffer contents are overwritten before playback is actually started or while playback is in progress, the contents are not played back correctly. Make sure that the sound output-only buffer is at least a double buffer so that the buffer contents immediately after sound is registered are not overwritten.

Example:

```
/* Let aPcmBuffer be a double buffer */
int bufferId = 0;

result = sceSasCore(aPcmBuffer [bufferId]);
if (SCE_SAS_FAILED(result)) {
        /* Audio data creation failed */
}

/* Output the 16 bit PCM STEREO data of aPcmBuffer [buffereId] to portId. */
result = sceAudioOutOutput(portId, aPcmBuffer[buffereId]);
if (result != SCE_OK) {
```

```
                /* Output failed */
        }

        /* Buffer switching */
        bufferId ^= 1;
```

8 Precautions

Thread Priority Settings

Make the priority of the sound processing thread that uses a function such as an audio output function to output PCM sound that was generated by `sceSasCore()` as high as possible. As long as there is no specific problem, setting `SCE_KERNEL_HIGHEST_PRIORITY_USER` is recommended. If the priority of the sound processing thread is low, a phenomenon may occur such as the generated sound being mixed with noise or the generated sound being intermittent or not even being played back at all.

To specify a high priority, the CPU affinity may need to be set properly. For details, refer to the "Kernel Overview" document.

When Raising the Pitch

When generating sound by specifying a high pitch, more phoneme samples must be processed, thereby increasing the CPU and memory band load. Therefore, if the pitch is raised for many of the sound-producing voices, the playback sound could become intermittent. In this case, try not to raise the pitch very much (prepare a phoneme sample that is recorded with increasing pitch) or to take measures, such as reducing the number of simultaneous produced sounds.

Parameter Update Timing

SAS uses batch processing to update internal parameters within the synthesis function (this function is called internally when a function such as `sceSasCore()` is called). As a result, the following are examples of the kinds of phenomena that occur.

When `sceSasSetVolume()` is used to change the volume, the volume will not change immediately

After the `sceSasSetVolume()` function is called, the volume will not be changed until the next time the synthesis function is called. The volume of the sound occurring at the instant `sceSasSetVolume()` is called will not change.

This phenomenon is not limited to the volume setting, but also occurs for all parameters such as key on, key off, and envelope shape changes.

The value that is returned by the `sceSasGetEndState()` function is not immediately updated after a key on or key off function (`sceSasSetKeyOn()` or `sceSasSetKeyOff()` function) is called

The `sceSasGetEndState()` function is used to get the sound generation end flags for all voices, however, the flags are updated only once at the moment when the synthesis function call ends. Therefore, when multiple key on or key off operations are performed for the same voice before the synthesis function is called, you cannot use the end flags obtained from `sceSasGetEndState()` to verify that a voice is in an inactive state.

Termination Processing

SAS internally operates mutexes for exclusive processing.

Therefore, if the thread that is using SAS is forcibly deleted by using `sceKernelDeleteThread()` before the thread has transitioned to DORMANT state, then depending on the timing, the thread may end up getting terminated and deleted while it is still holding internal resources. As a result, there is a risk that another library, including SAS, may no longer operate properly.

To terminate a thread, make sure the following actions are taken (the samples that use SAS are also constructed in this way):

- The side that requests the thread to terminate uses a shared variable or event flag to communicate its request to the thread.
- The thread that is to be terminated periodically monitors this flag to determine whether a termination request was issued. If a request was issued, the thread first verifies that SAS is not currently running and then it calls `sceKernelExitThread()` to terminate itself.

Restrictions

The following are the current restrictions.

- Currently, SAS does not use the Codec Engine but operates only with ARM.
- ATRAC3™ and other compression codecs are currently not supported.