Revolution

Audio Library

Version 1.00

Audio Library

Version 1.00

Contents

Figures

Tables

Revision History

Revision No.	Date Revised	Items (Chapter)	Description
22-Mar-2006	3/22/2006	-	First release by Nintendo of America Inc.

1 Introduction

The AX library is a thin, low-level interface to the Revolution audio subsystem. The AX library abstracts the details of managing the audio DSP and associated audio hardware into a convenient library. The AX library has the following features.

- Allows user applications to allocate audio voices, and it provides access to the parameters that describe each voice.
- Automatically generates DSP commands for creating audio as dictated by changes to the aforementioned voice parameters.
- Manages voice contention and audio system resources.
- Provides efficient, low-level control of the audio subsystem so that developers may create their own audio applications without worrying about hardware interfaces.

1.1 Terms

The terms in Table 1-1 are used throughout this document.

Table 1-1 AX Library Terms

Term	Description
AI	Audio Interface. Provides the data path between the major components of the audio subsystem hardware. Specifically, the AI routes data between the DSP and the external DAC.
SRC	Sample Rate Converter. Interpolates audio data samples to change the sample rate of specified data.
Volume	In this context, a multiplier applied to the samples of an audio signal to modulate the signal's amplitude.
Volume Envelope	A fixed change to volume over time.

2 AX Processing Model

2.1 System Components

The following subsections provide a brief description of the Revolution audio subsystem.

2.1.1 AI

The Revolution Audio Interface (AI) routes audio data from the DSP and optical disc streaming interfaces to the external 48 KHz stereo DAC. The AI also includes a pair of high-quality 32 KHz-to-48 KHz sample rate converters; these convert output from the DSP and optical disc streaming interfaces (if needed) prior to mixing.

The AI's interface to the DSP consists of a programmable DMA which transfers data from main memory to a FIFO buffer. This DMA is referred to as the AI-FIFO DMA and is managed by the AX library.

For more information about the Audio Interface, including its API functions, refer to the **Audio Interface** pages in the Audio section of the Revolution Function Reference Manual (HTML).

2.1.2 DSP

The Revolution audio system includes a proprietary digital signal processor (DSP). Clocked at one half the speed of the CPU, the DSP generates audio data as dictated by the voice parameters maintained by AX. Specifically, the DSP is responsible for:

- Retrieving samples from main memory.
- Sample rate conversion.
- ADPCM decompression (if necessary).
- Applying a volume envelope.
- Applying AUX-send-return effects.
- Mixing.

Note: The microcode application used in the DSP is proprietary and currently not available for review and/or modification.

2.2 Processing Flow

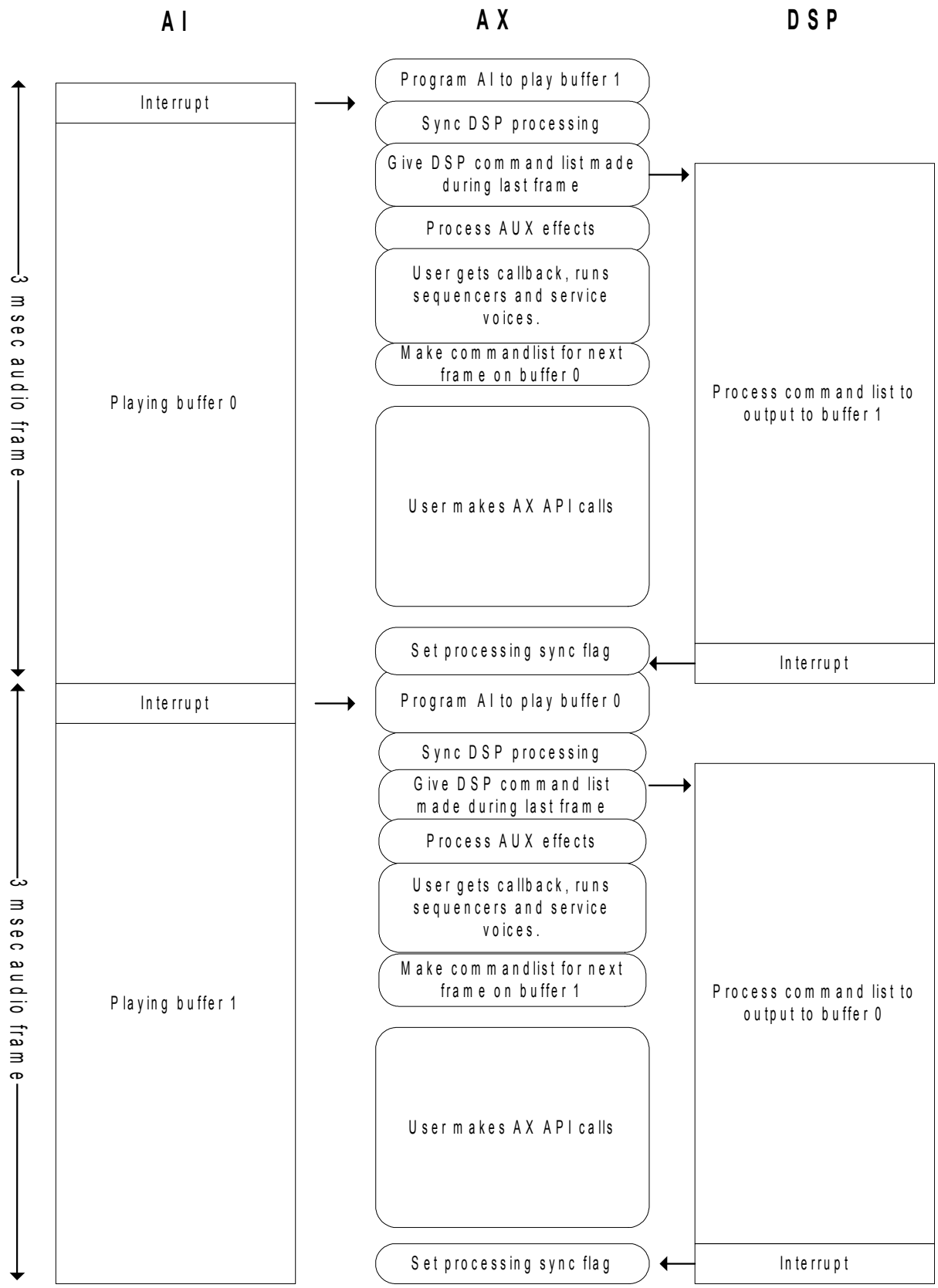
The AI-FIFO DMA interrupt drives AX. Upon receipt of this interrupt, AX generates a command list for the DSP to produce a three-millisecond "frame" of audio data. AX buffers two frames of audio data and directs the AI-FIFO DMA to alternate between the frame buffers every three milliseconds. (It is also possible to have a triple buffer structure, which buffers three frames of audio data by using the `AXInitEx` function.)

AX generates a command list as dictated by the state of voice parameters, which are maintained in main memory and accessible by the user audio application. The DSP generates audio per the commands in the command list and transfers the data to main memory for consumption by the AI-FIFO DMA.

For more information about the AI-FIFO DMA, refer to the **Audio Interface** pages in the Audio section of the Revolution Function Reference Manual (HTML).

Figure 2-1 offers an overview of the AX processing model.

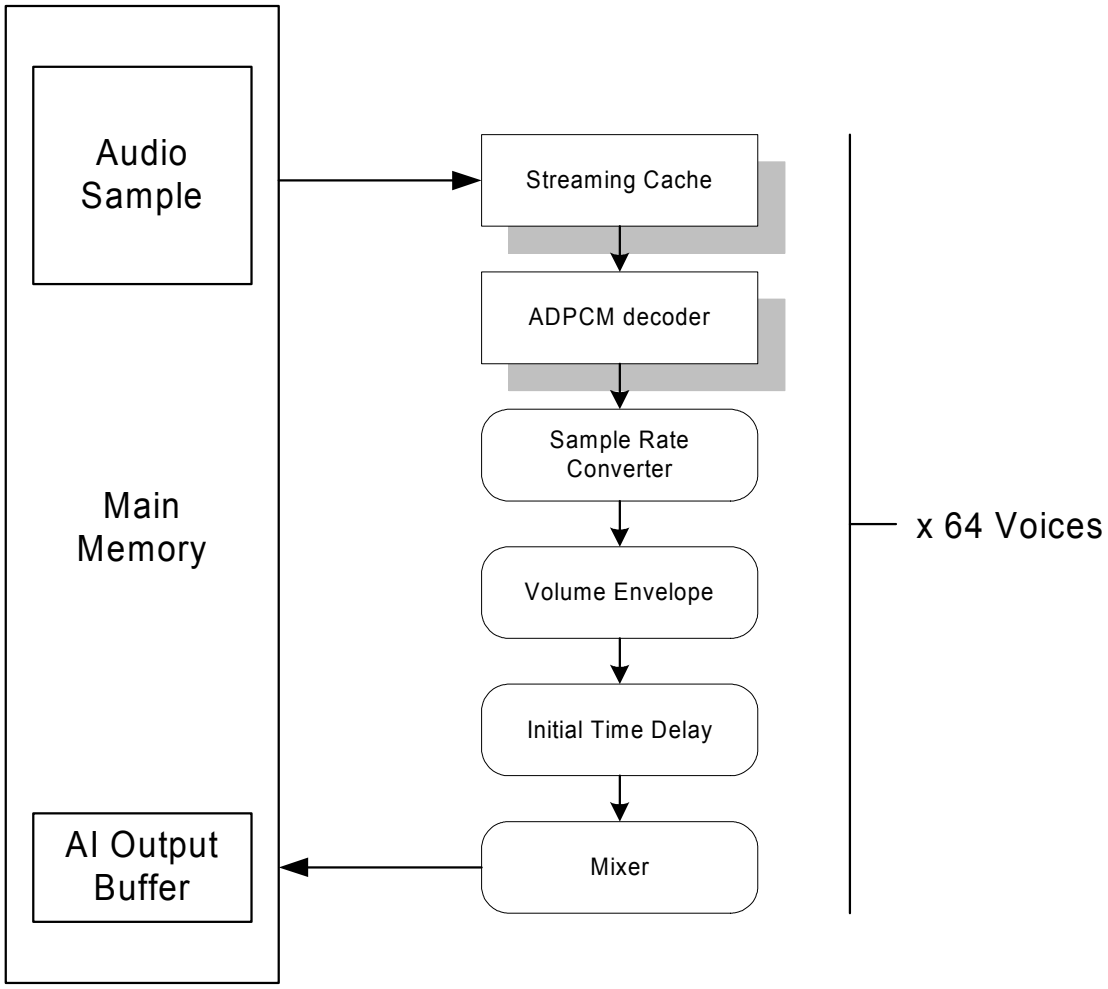
Figure 2-1 AX Logical Flow



2.3 Voice Processing Pipeline

Figure 2-2 presents an overview of the DSP data flow and processing tasks for each voice.

Figure 2-2 Voice processing flow



2.3.1 Streaming Cache

The streaming cache is a dedicated memory interface that provides the DSP with fast linear access to main memory.

It features automatic address-generation and allows the DSP to access main memory as nibbles, bytes, or (16-bit) words.

The streaming cache also provides automatic loop-address-generation, thus requiring programming of current, loop-start, and loop-end address parameters. When the current address exceeds loop-end, the streaming cache will automatically revert to the specified loop-start address.

2.3.2 ADPCM Decoder

The DSP contains a hardware decoder coupled to the streaming cache. This decoder accelerates decompression of ADPCM samples and conversion of 8-bit PCM samples to 16-bit.

The decoder may also be programmed to pass **raw** 16-bit PCM data unchanged.

Note: The decoder and streaming cache work together to handle looped ADPCM samples. Specifically, the ADPCM context is stored within the voice parameter block automatically and retrieved as needed at loop events.

2.3.3 Sample Rate Converter (SRC)

Sample rate conversion is handled by the DSP microcode. Three types of SRC are provided:

- Interpolation w/ 4-tap FIR filter.
- Linear interpolation, no filter.
- Conversion bypass (no rate change).

The 4-tap FIR filter supports three different coefficient tables to provide three different frequency responses (8KHz, 12KHz, and 16KHz).

2.3.4 Volume Envelope

The DSP applies the volume envelope to sample data as specified by the multiplier and delta in the voice parameter block. Each sample is multiplied by the multiplier; the delta is added to the multiplier after each multiplication.

The volume envelope may represent the sum of the volume fader, ADSR envelope, and tremolo algorithms (as computed by the user audio application).

The DSP does not clamp the multiplier and delta computations; rather, the host audio application is responsible for ensuring that these calculations will remain within bounds.

2.3.5 Filtering

(Left blank because specifications are undetermined.)

2.3.6 Initial Time Delay (ITD)

The DSP mixer can introduce up to one millisecond of phase shift for the left and right channels of the main bus.

The ITD is programmed with a shift value (in samples) for each channel. The shift is not introduced immediately; instead, the DSP **crawls** towards the target value to prevent popping or rate conversion artifacts.

2.3.7 Mixing

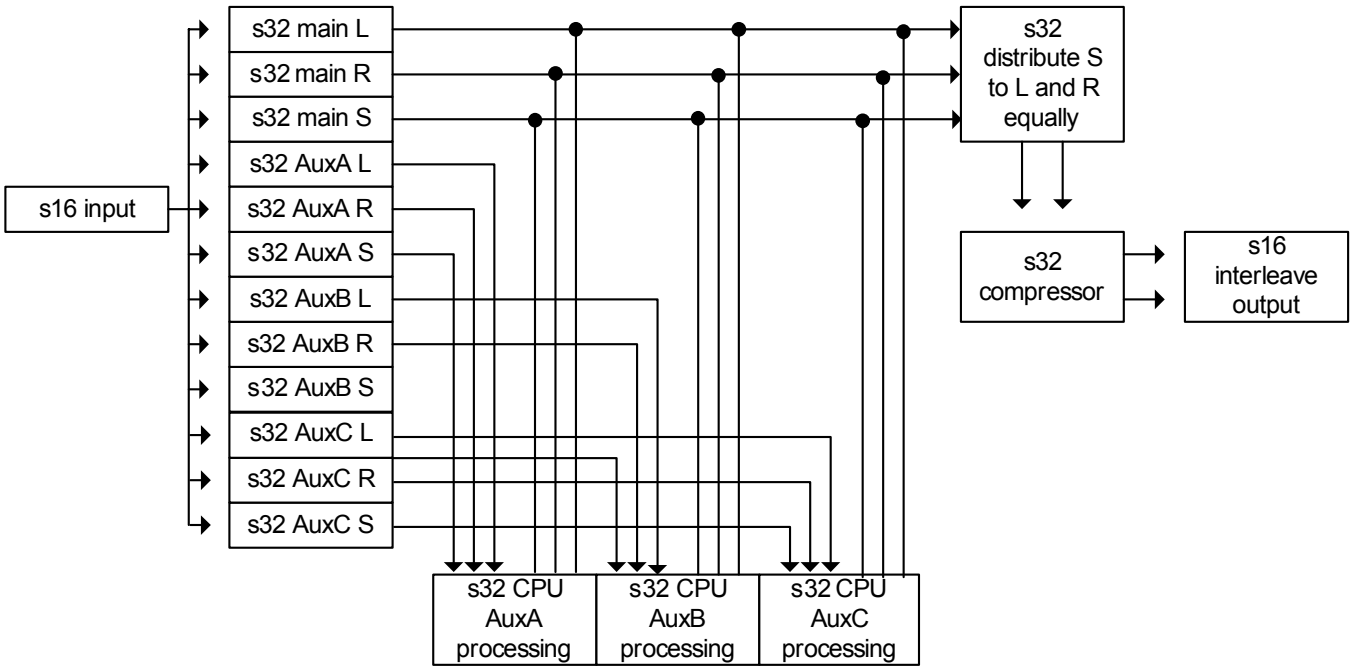
Several mixing modes are available for AX. These are described in the following subsections.

Notes:

- L: Left channel
- R: Right channel
- S: Surround channel

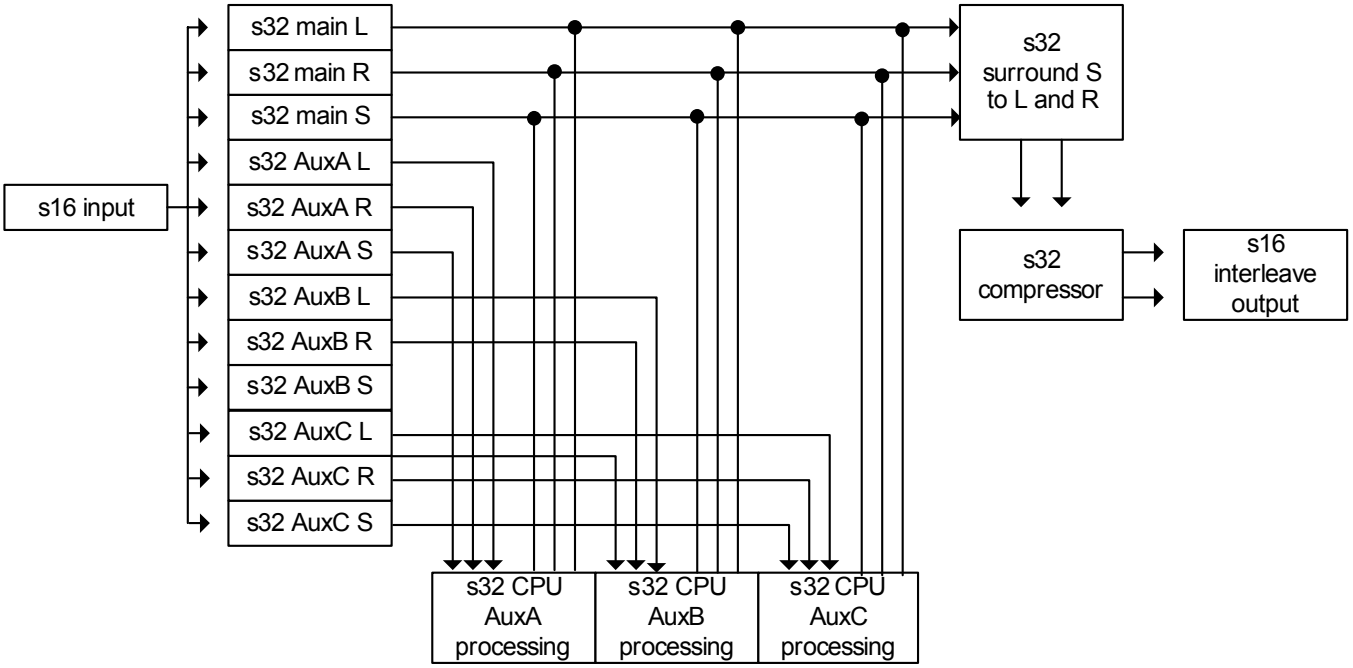
2.3.7.1 Stereo Mixing

Figure 2-3 Stereo Mixing



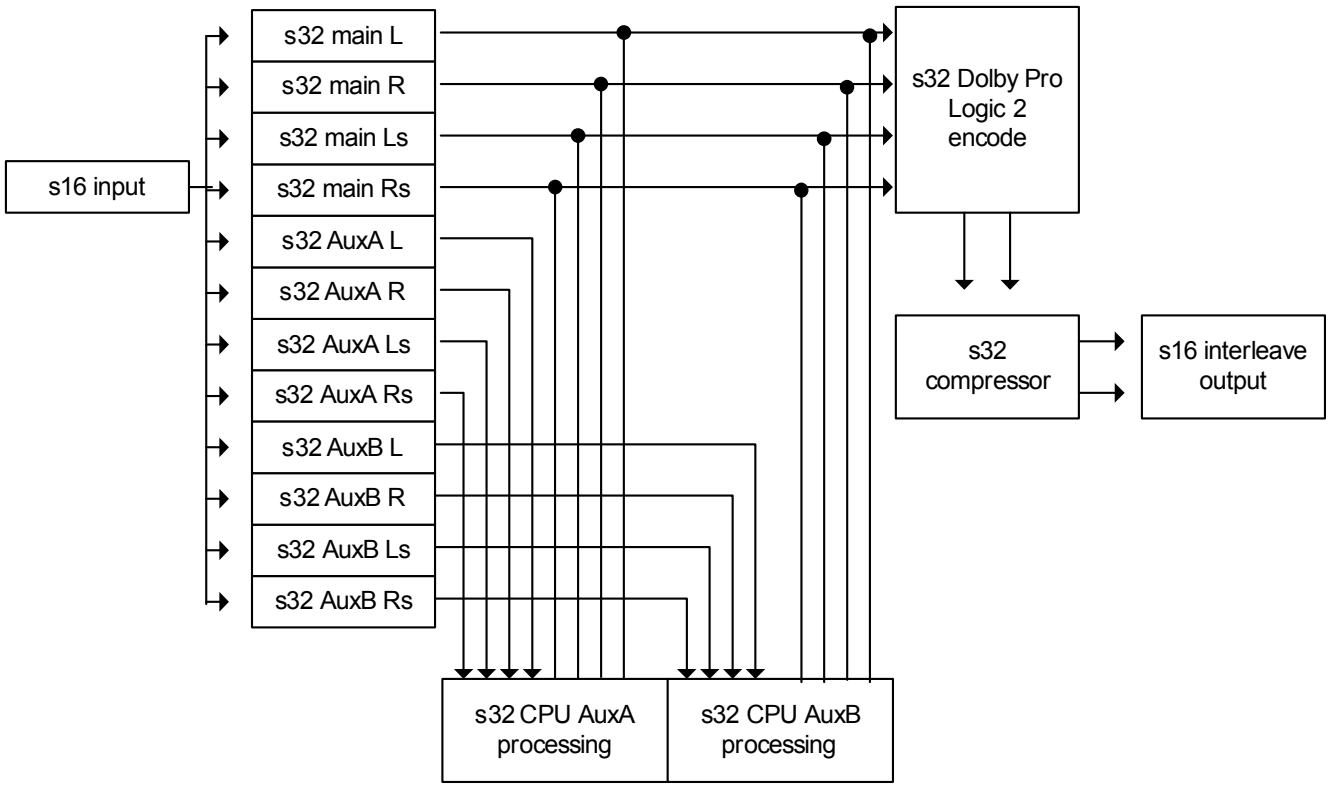
2.3.7.2 Surround Mixing

Figure 2-4 Surround Mixing



2.3.7.3 Dolby Pro Logic 2 Mixing

Figure 2-5 Dolby Pro Logic 2 Mixing



3 Application Notes

3.1 Voice Acquisition

AX handles voice allocation internally so that automatic **de-popping** occurs for voices being dropped due to resource conflicts. A call to `AXAcquireVoice` will cause AX to attempt to acquire a voice for use in the following fashion:

- Acquire a free voice.
- Acquire the oldest or lowest priority voice (i.e., lower in priority than the user-specified voice).
- Acquire no voice.

AX may also drop voices during processing if there are not enough DSP cycles to process all the voices. In this event, AX will drop as many of the oldest or lowest priority voices as required. A voice acquired with `AX_PRIORITY_NODROP` may be dropped in this fashion.

Table 3-2below offers suggestions for voice priority usage. Some may be more useful than others in regard to specific game development.

Table 3-2 Voice Acquisition Priority

User Application	Priority
Voice and Music Streams	Typically cannot be dropped; therefore, they should be acquired with the highest priority (`AX_PRIORITY_NODROP`).
Sound Effects	High in priority, but some types of effects may become lower in priority as time elapses. The user application may acquire voices at high priority and re-assign them internally to effects; alternatively, it can set them to a low priority once they are no longer important. Varying degrees of lower priorities may also be used; e.g., a machine gun might use the same voice over and over, while other sound effects (such as environment sounds) may be set to an increasing lower priority as they grow further away and become less important.
MIDI Synthesizer	Should acquire voices with high priority, but priority should not be higher than the more important sound effects and voice/music streams. Once the note is played, set the priority for that voice lower so that any new notes may be played. Any notes entering ADSR release stage can be set to low priority to ensure that they will be acquired first in the event of voice contention; e.g., acquire voices with priority 16. Once the note is on, set the voice priority to 15. When the voice is set to release, set priority to 1.

Because voices may be dropped at any time, the user application should implement a callback function to clear any references made to a voice. This will ensure that the no more servicing occurs for that voice once it has been reacquired for use by the same or another application.

3.2 Buffer-Addressing

Here are a few things to keep in mind when handling buffer-addressing:

- Parameter blocks sample buffer-addressing is in 16-bit word, byte, or nibble modes for 16-bit PCM, 8-bit PCM, and ADPCM buffers, respectively.
- The current address, loop address, and end address include the first or last sample played. For example, a 4 KB circular buffer starting at 0 should be addressed with the current address and loop address at 0, and the end address at 4 KB – 1.
- ADPCM buffers must start on an 8-byte boundary in main memory.
- Addressing for ADPCM buffers cannot land within the 2-nibble frame header for any 8-byte frame; AX will ASSERT this on DEBUG builds.
- The user application must program the initial ADPCM predictor and scale before starting a voice.

3.3 Optimization

If your application uses frequent `AXSetVoiceX` calls at runtime, you can optimize code by printing to the parameter block directly. Here are some helpful guidelines:

- Review the **AX Voice Parameter Blocks** pages in the Audio section of the Revolution Function Reference Manual (HTML) for a detailed description of precedence for the voice parameter block sync flag.
- Disable interrupts when adjusting voice parameters.
- You may enable interrupts when adjusting voice parameters from within the AX user callback.

3.4 Voice Indexing

The `AXVPB` data structure includes an **index** member. This index is assigned by AX to each voice during initialization. This index is intended to aid application voice referencing.

- Applications may implement a table of AX_MAX_VOICES entries to store voice referencing.
- Service acquired voices by referencing table entry at [voice $\rightarrow$ index].
- Do not write to the [voice $\rightarrow$ index] table entry.

3.5 Zero Buffer

The **zero buffer** is a contiguous block of at least 256 bytes in main memory. As you may guess, each byte must be set to zero. Furthermore, the buffer must be 32-byte aligned in main memory and must be a multiple of 32 bytes in length. The buffer is required for all DSP-generated audio.

The buffer is part of a sample address optimization mechanism used by DSP for reducing processing overhead during voice mixing. In essence, the DSP checks whether the voice has reached the end of the sound effect every 1 msec rather than every **sample**. When the sound effect reaches its end, the sample address engine harmlessly takes samples from the zero buffer until the DSP can turn off the voice.

Because voice **acquisition** only happens at the (3 msec) frame boundaries, there is no difficulty with the 1 msec precision for voice termination.

This optimization saves several million DSP cycles per second.