Revolution Audio Library

Version 1.01

The contents in this document are highly confidential and should be handled accordingly.

RVL-06-0031-001-C
Released: October 22, 2007

Contents

Figures

Tables

Revision History

Version	Date Revised	Item	Description
Version 1.01	2006/11/24	2.1.2	Added controller speaker processing.
		2.3	Updated Figure 2.
		2.3.5	Added a description of the IIR filter.
		2.3.8	Added a description of controller speaker processing.
		3.1	Revised the description of AX_PRIORITY_NDROP.
		3.5	Deleted the zero buffer.
Version 1.00	2006/03/01	-	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 provides access to the parameters that describe each voice.
- Automatically generates DSP commands for creating audio as dictated by changes to these voice parameters.
- Manages voice contention and audio system resources.
- Provides efficient, low-level control of the audio subsystem so that you can create your 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 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 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 32-48 KHz sample rate converters. These convert output from the DSP and optical disc streaming interfaces (if needed) prior to mixing.

The AI 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, see “Audio Interface (AI)” in the audio system section of the *Revolution Function Reference Manual* (HTML).

2.1.2 DSP

The Revolution audio system includes a proprietary digital signal processor (DSP). The DSP generates audio data as dictated by the voice parameters maintained by the AX library. Specifically, the DSP carries out the following processing:

- Retrieving sampled data from main memory (streaming cache)
- ADPCM decoding
- Sample rate conversion
- Applying a volume envelope
- Applying an IIR filter
- Mixing
- Generating audio data for the controller speaker
- Applying AUX-send-receive effects

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

2.2 Processing Flow

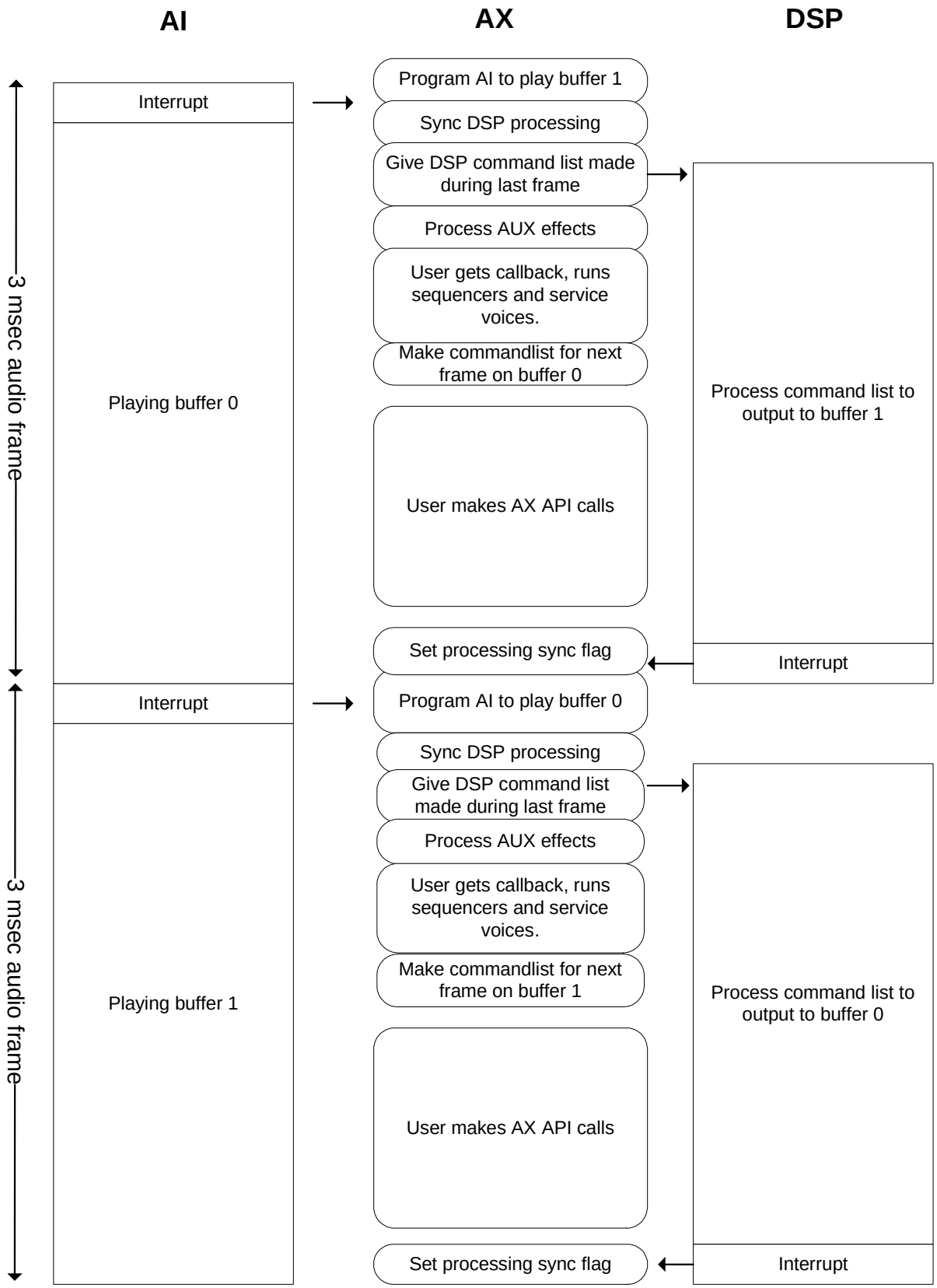
The AI-FIFO DMA interrupt drives AX. Upon receipt of this interrupt, AX generates a command list for the DSP to produce a three-millisecond “frame” of audio data. AX buffers two frames of audio data and directs the AI-FIFO DMA to alternate between the frame buffers every 3 ms. (It is also possible to 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 according to 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, see “**Audio Interface (AI)**” in the audio system section of the *Revolution Function Reference Manual* (HTML).

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

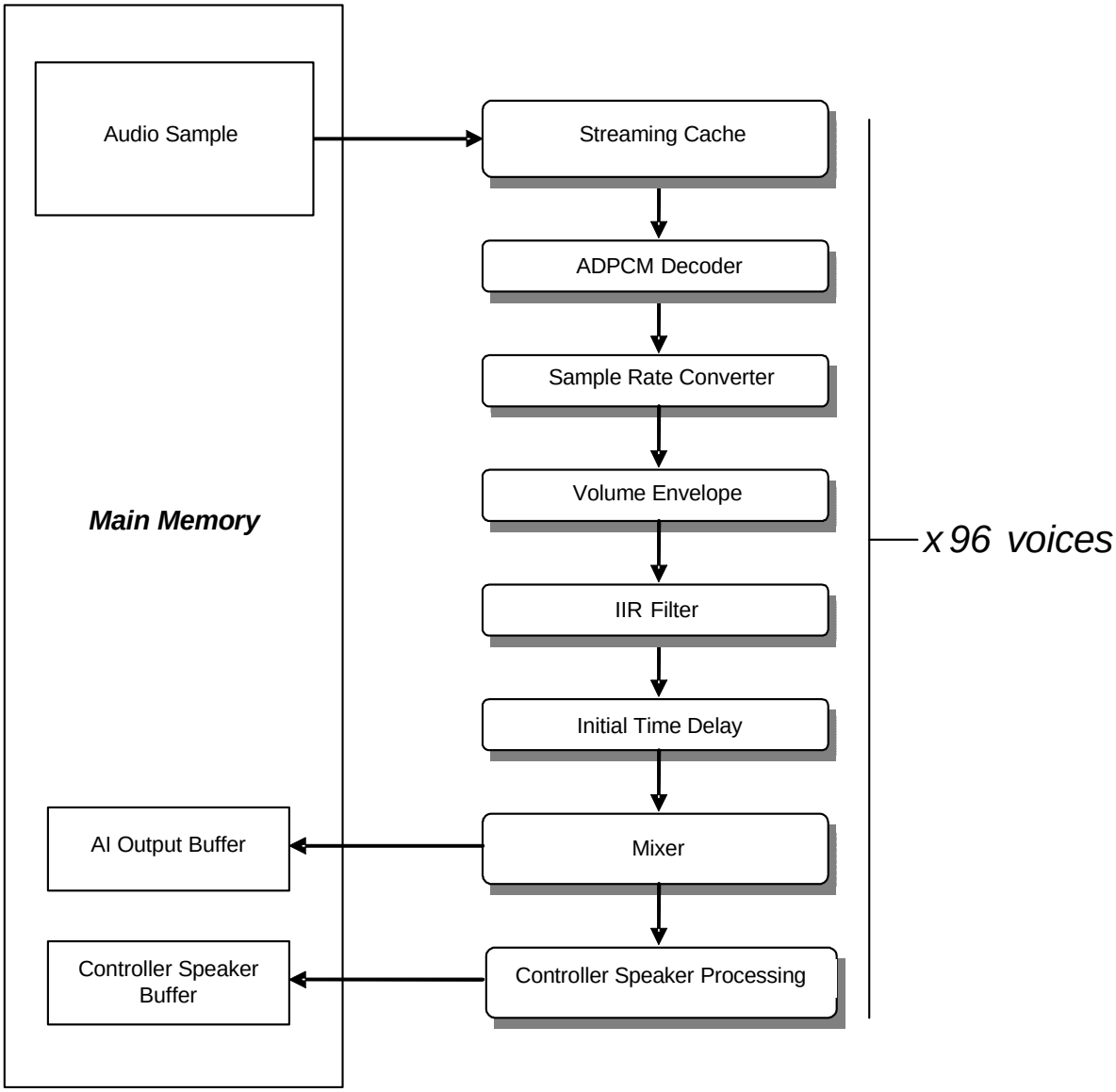
Figure 2–1 AX Logical Flow



2.3 Voice Processing Pipeline

Figure 2–2 represents the process flow for each voice handled in the DSP.

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. Each of the current address, loop-start address, and loop-end address parameters must be set for the DSP before utilizing the streaming cache. When the current address reaches the end address, the streaming cache automatically reverts 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.

The decoder (and streaming cache) supports looped ADPCM data. The DSP accesses the ADPCM context stored within the voice parameter block when the decoder generates a loop event. The ADPCM context is automatically stored in data when data is created. The application must configure this ADPCM context within the voice parameter block when voices are initialized.

2.3.3 Sample Rate Converter (SRC)

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

- Interpolation with 4-tap FIR filter
- Linear interpolation without a filter
- Conversion bypass with no rate change

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

2.3.4 Volume Envelope

The DSP applies the volume envelope to sampled data, using the current volume level in the voice parameter block and the volume difference value of each sample.

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 perform clamping while calculating the volume envelope. The audio application must guarantee that these calculations remain within determined bounds.

2.3.5 IIR Filter

The DSP can apply an IIR filter to sampled data. There are two types of IIR filter: a simple filter that performs light processing and features gentle cutoffs and a biquad filter that performs more intense processing (three times the load of the simple filter) and features steep cutoffs.

It is possible not only to select which filter to use for each voice and whether to turn it on or off, but also to modify the filter coefficient. It is also possible to apply both the simple filter and biquad filter to the same voice. (When this is done, the simple filter first is applied first, then the biquad filter.)

2.3.6 Initial Time Delay (ITD)

The DSP mixer can introduce up to 1 ms of phase shift for the left and right channels of the main bus.

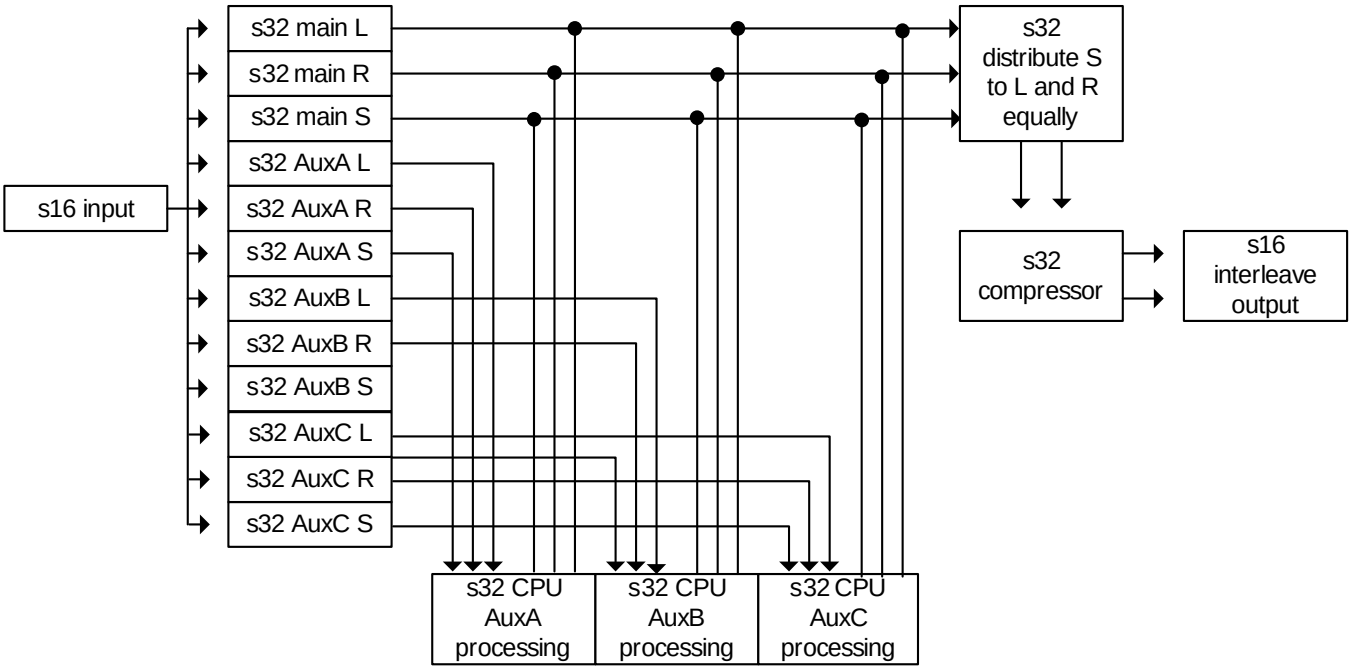
The ITD is programmed with a shift value (in samples) for each channel. The shift is not introduced immediately; instead, the DSP **crawls** towards the target value to prevent popping or rate-conversion artifacts.

2.3.7 Mixing

Several mixing modes are available for AX. These are described in the following subsections. In the following diagrams, L indicates the left channel, R, the right channel, and S, the surround channel

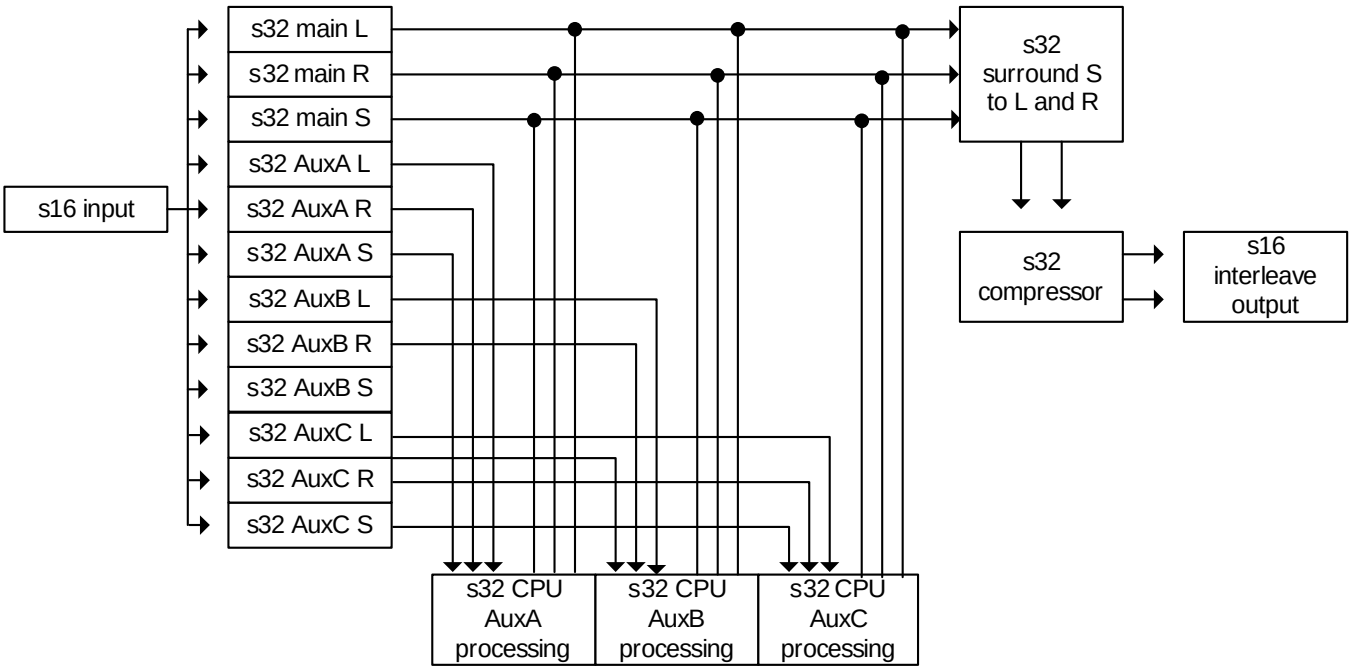
2.3.7.1 Stereo Mixing

Figure 2–3 Stereo Mixing



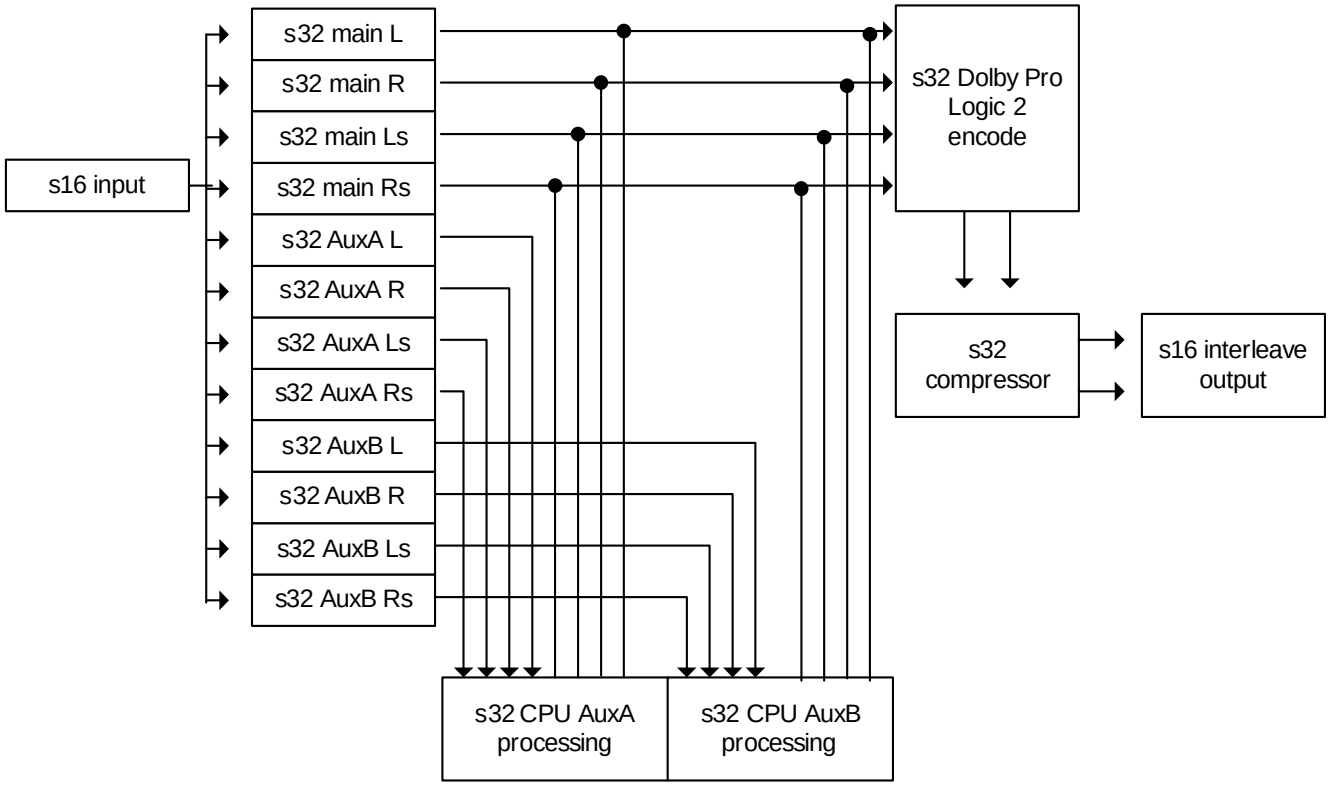
2.3.7.2 Surround Mixing

Figure 2–4 Surround Mixing



2.3.7.3 Dolby Pro Logic 2 Mixing

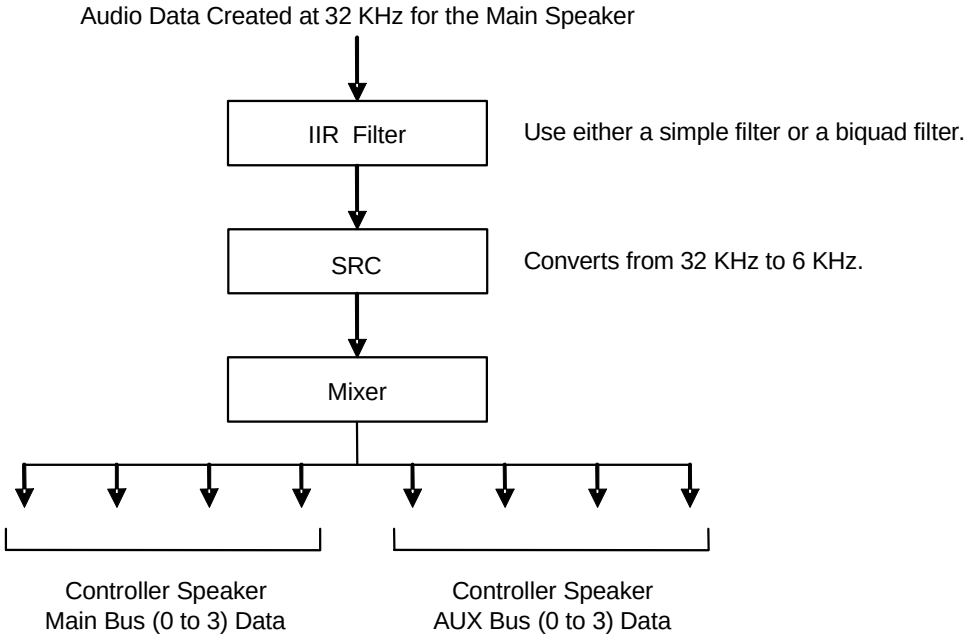
Figure 2–5 Dolby Pro Logic 2 Mixing



2.3.8 Controller Speaker Processing

The DSP carries out the following processing for the controller speaker.

Figure 2–6 Controller Speaker Processing



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 AX`AcquireVoice` causes AX to attempt to acquire a voice for use in the following fashion:

- Acquire a free voice
- Re-acquire the oldest voice with the lowest priority (of those voices having priority lower than new voices)
- Not acquire a voice (cannot be acquired)

AX drops voices during processing if the number of DSP cycles required to process all the voices exceeds the number of DSP cycles available. In this case, AX drops as many of the oldest and lowest priority voices as required.

Although a voice whose priority has been specified as `AX_PRIORITY_NODROP` is not dropped due to resource conflicts, it will be dropped when the number of DSP cycles required to process voices exceeds the number of available DSP cycles.

Table 3–1 offers suggestions for voice priority usage. Some may be more useful than others in game development.

Table 3–1 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 in order of high priority and, internally within effects, re-assign them to a low priority once they are no longer important. Applications may also be used to set varying degrees of lower priorities; for example, 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 case of voice contention; for example, 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.

Voices may be dropped at any time. Once a voice is dropped, the application must make sure that the voice is not referenced inappropriately. Be sure to handle this by clearing the pointer to the voice in the callback function specified by the argument to the AX`AcquireVoice` function. This ensures that the voice can be used safely in the same sound routine or in mutually independent sound routines.

3.2 Buffer-Addressing

Keep the following facts in mind when specifying a buffer address for sampled data in the voice parameter block:

- When setting the address, you must change the address unit according to the format of the data. If the sampled data is 16-bit PCM, specify the buffer address in WORDs (16-bit units). If the format is 8-bit PCM, specify the address in bytes, and if the format is ADPCM, specify the address in nibbles (4-bit units).
- Specify the address in which sampled data to be played is stored for the current address, loop-start address, and loop-end address. For example, if a 4096-byte looped 8-bit PCM data sample is stored in a buffer starting at address 0, specify zero for the current address and loop-start address, and 4095 for the loop-end address.
- ADPCM data has a frame length of 8 bytes, where the first byte represents the frame header, and the remaining 7 bytes represent data. When loading ADPCM data into a buffer, the frame header of each ADPCM frame must be aligned at an 8-byte boundary.
- Avoid the ADPCM frame header when setting the address of the ADPCM buffer. In other words, it is impossible to set the first byte (2 nibbles) of an 8-byte (16-nibble) frame for each address.
- Applications must set the ADPCM predictor and scale during voice initialization.

3.3 Optimization

If your application uses frequent calls at runtime to AX`SetVoiceXXX` to set voice parameters, you can optimize code by directly editing the parameter block. Here are some helpful guidelines:

- When you have changed the voice parameter block, you must set the voice parameter block sync flag (`axvbp.sync`) appropriately. For a detailed description of the sync flag, see `axvpb.sync` in “Audio Library (AX)” in the Audio section of the *Revolution Function Reference Manual*.
- Disable interrupts when changing the voice parameter block. However, interrupts can be enabled when changing the voice parameter block from an 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 aids application voice referencing.

- Applications may implement a table of AX_`MAX_VOICES` entries to store voice referencing
- Service acquired voices by referencing the table entry at [`voice` → `index`]
- Do not write to the [`voice` → `index`] table entry