T-Engine Forum
Specification

TEF040-S213-01.00.00/en
September 29, 2003

Standard Audio Device Driver Specification

Number:TEF040-S213-01.00.00/en
Title: Standard Audio Device Driver Specification
Status: [ ] Working Draft, [ ] Final Draft for Voting, [*] Standard
Date: July 22, 2003 First Edited (Takeshi Aida, Faith, Inc.)
August 5, 2003 Updated to 00.00.02 (revised Section 2.7 and the figures)
September 29, 2003 Voted

Contents

1 Introduction

This specification defines the standard API for a device driver that enables audio IO with T-Engine.

The driver provides audio codec control. In a minimal configuration, the driver is designed to support hardware equipped with one audio input or output system or both. It was also designed considering expandability for hardware with a digital IO interface and analog volume control.

Two methods of supporting hardware expansion are assumed: an expanded audio codec can be supported with more driver units, and multiple implementations of DAC and ADC can be supported with more driver subunits.

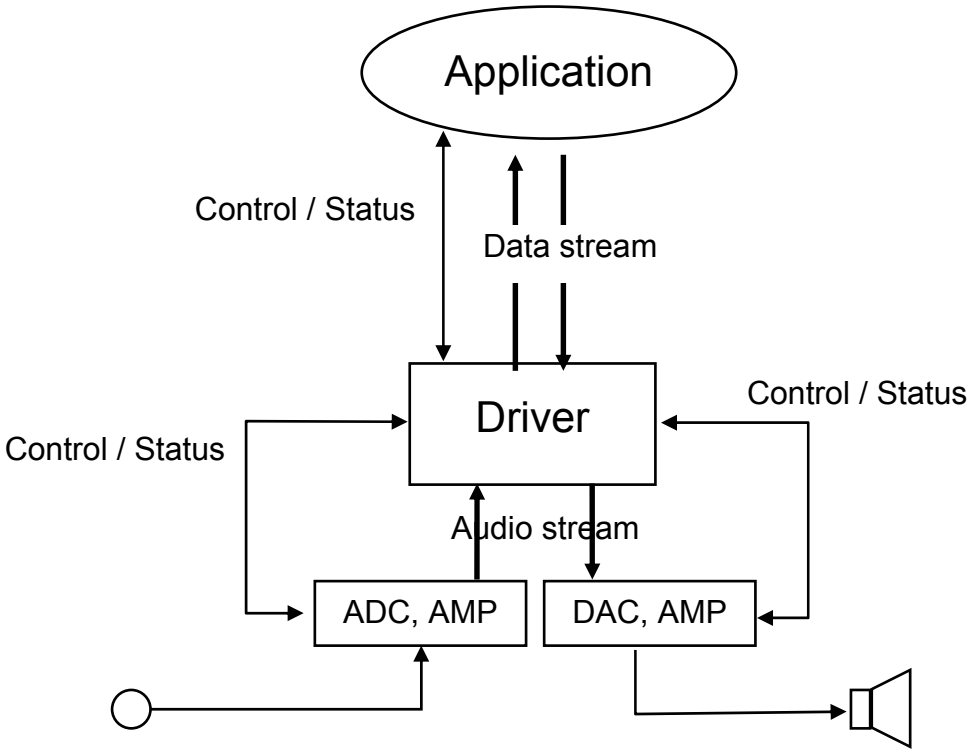

Figure 1. Schematic diagram

2 Driver

2.1 Implementation Requirements

- The block size of audio device-specific data is defined in AUDIO_DEVBLKSIZE [byte].
- The subunit ID is assigned by the DAC or ADC implementation. The subunit number starts with 0.
- The driver must support the asynchronous read/write mechanisms tk_wri_dev(), k_rea_dev(), and tk_wai_dev() for device-specific data.
- Queuing is enabled up to AUDIO_MAXREQQ times for read and write in asynchronous access requests to device-specific data. In other words, if AUDIO_MAXREQQ=2, even when the queue becomes full of write requests as depicted in Figure 2, if read requests are issued the driver must receive them without any wait. This is not required, however, if the hardware does not support full-duplex operation.

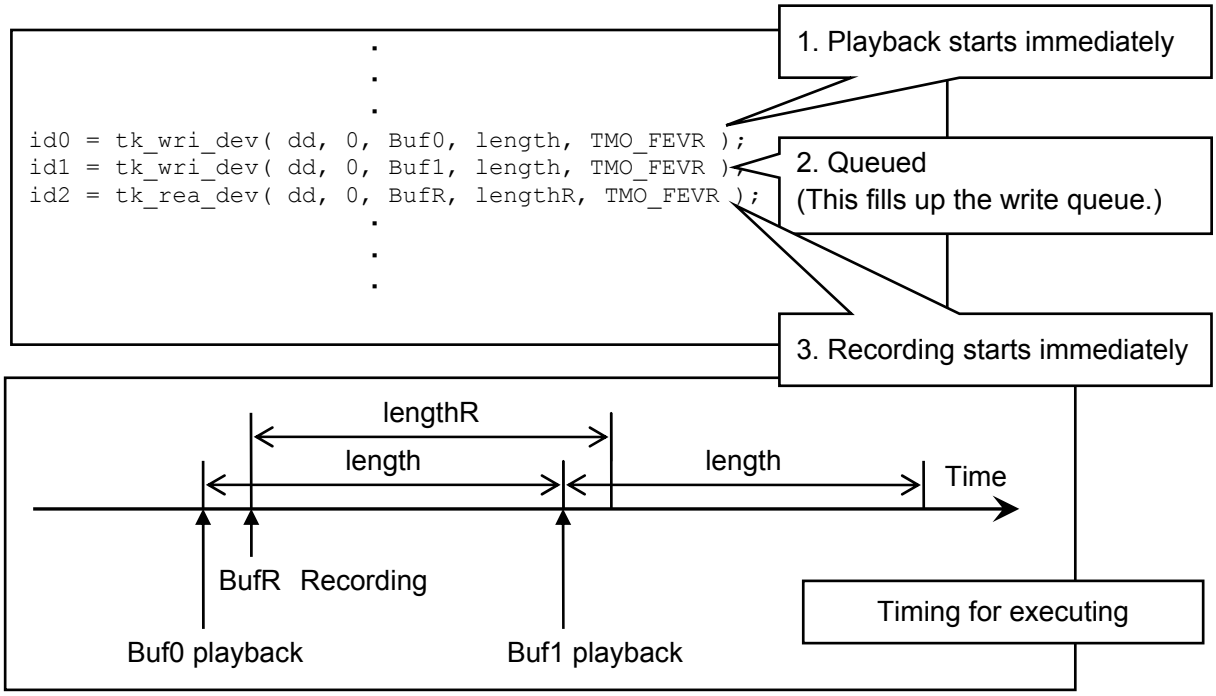

Figure 2. Asynchronous access requests and queuing

- All access requests for attribute data must be immediately executed. In addition, access to attribute data is provided regardless of the queuing status of access requests for device-specific data. In other words, if AUDIO_MAXREQQ=2, even when the queue becomes full of write requests for device-specific data as depicted in Figure 3, if write requests for attribute data DN_AUDIO_MIXERSETOUTPUTVOL are issued, the driver must receive them without any wait for the write to DN_AUDIO_MIXERSETOUTPUTVOL.

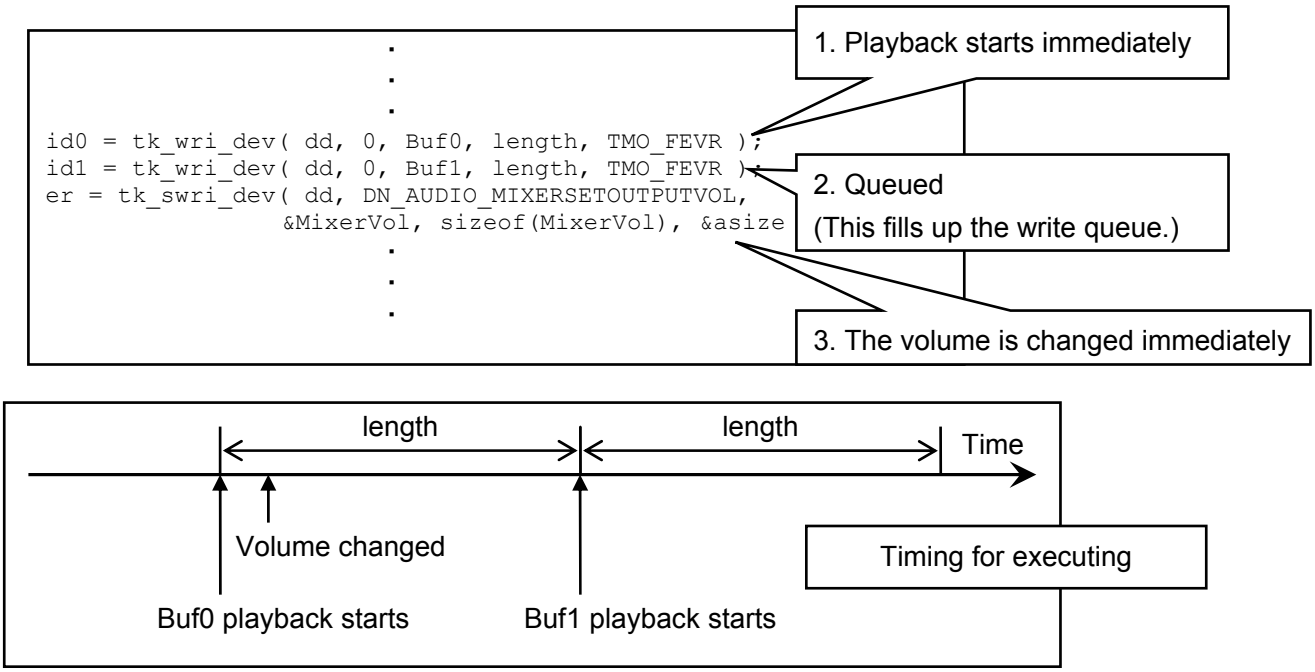

Figure 3. Asynchronous access requests and access to attribute data

- DMA transfer is assumed for this explanation. For platforms not supporting DMA transfer, the implementation is platform-dependent.

- Items herein that are not described in detail should conform to provisions of T-Engine standards.

2.2 ID tk_opn_dev(UB *devnm, UINT omode)

Reserves the device and establishes standby.

Argument

devnm

Pointer to the device name string
The device name comprises "audio" to distinguish the type, followed by the unit, represented by a letter from a to z, and the subunit, represented by a number. The first unit is designated by the device name "audioa0" if the subunit is 0.

omode

Designates options.

TD_WRITE	Enables processing for audio playback
TD_READ	Enables processing for audio recording
TD_NOLOCK	No locking of the specific data buffer by the driver

Whether or not simultaneous setting of TD_WRITE and TD_READ (full-duplex operation) is available is hardware-dependent. However, read and write of attribute data (start < 0) is supported regardless of the omode designation.
With tk_wri_dev() and tk_rea_dev(), the driver locks and unlocks the relevant buffer (making them resident or non-resident) for specific data before and after access. If TD_NOLOCK is designated here, this is not performed.

2.3 ER tk_cls_dev(ID dd, UINT option)

Stops the device and releases it.
Interrupts active read or write operations and cancels all queued requests.
If the driver message buffer (see Section 2.5.2) is registered by DN_AUDIO_REGISTERMSGBUF, this releases it.

2.4 ID tk_wai_dev(ID dd, ID reqid, INT *asize, ER *ioer, TMO tmout)

Waits for read and write requests indicated by reqid to be completed.
If it ends in error, *asize is undefined.

2.5 ID tk_wri_dev(ID dd, INT start, VP buf, INT size, TMO tmout)
ER tk_swri_dev(ID dd, INT start, VP buf, INT size, INT *asize)

2.5.1 Audio playback

Designates the start address and number of blocks of the data for playback and starts playback.
If requests are queued through asynchronous processing, the driver must enter processing for the next request continuously after the current request is complete without pause.
A buffer must not be released once it has issued a write request until that request is complete.
If the driver message buffer (see Section 2.5.2) is registered by DN_AUDIO_REGISTERMSGBUF, the driver issues a notification message immediately before and after buffer playback.

Data number `(start)`
DN_AUDIO_PLAYAUDIO (=0)

Argument

`buf`
The start address of the buffer where playback data is stored.
Depending on the hardware platform, the following limitations may occur.
(Examples of limitations)
Alignment required for each memory page
Must not be in task-specific space

`size`
The number of buffer blocks

2.5.2 Driver message buffer registration

Registers the driver-specific message buffer.
If the message buffer is already registered, the registered message buffer ID is returned without registration.
It may be difficult for the invoker to know the actual timing for processing read and write requests that are issued asynchronously. But through the message buffer registered here, notification of the start and end of the read or write can be received from the driver.
It is advisable that the timing when messages are issued coincide with the moment of playback and recording.
This message buffer is not registered by default.
The message from the driver is sent in the AudioMsgPacket structure.

```
typedef struct {
   ID id;
   VP buf;
   SYSTIM otm;
} AudioMsgPacket;
#define AUDIO_MSGPKTID_WRITESTART     0x0000
#define AUDIO_MSGPKTID_WRITECOMPLETE  0x0001
```

```
#define AUDIO_MSGPKTID_READSTART      0x0002
#define AUDIO_MSGPKTID_READCOMPLETE   0x0003
```

The message type is indicated in id. This corresponds to one of the following four types.

- AUDIO_MSGPKTID_WRITESTART The message is a notification of the start of playback
- AUDIO_MSGPKTID_WRITECOMPLETE The message is a notification of the end of playback
- AUDIO_MSGPKTID_READSTART The message is a notification of the start of recording
- AUDIO_MSGPKTID_READCOMPLETE The message is a notification of the end of recording

buf indicates the buffer start address (buf of tk_wri_dev(), tk_rea_dev()) of the read or write requests of the message source.
otm indicates the system up time when the message was issued (in the same way it can be retrieved with tk_get_otm()).

The driver has an internal status for notification if the message buffer is flooded.
Read and write for this internal status is done by setting the internal driver status (DN_AUDIO_SETSTATUS) or referring the internal driver status (DN_AUDIO_GETSTATUS).
No message is issued if the message buffer becomes full. A bit is set in AUDIO_STATUS_MBFFLOW in the internal status.

Data number `(start)`
DN_AUDIO_REGISTERMSGBUF

Argument
`buf`
Pointer to the ID type that stored the created message buffer to be registered
`size`
Must be sizeof(ID).
`Return Code`
The ID of the registered driver message buffer

2.5.3 Driver message buffer release

Releases the driver-specific message buffer.
If the message buffer is not registered, E_OBJ is returned.
buf and size are ignored.

Data number (start)
DN_AUDIO_UNREGISTERMSGBUF
Return Code
The ID of the released driver message buffer

2.5.4 Internal driver status setting

Sets the driver-specific internal status.
Details of the UW-type variable indicated by buf are copied to the internal driver status.

```
#define AUDIO_STATUS_MBFFLOW    0x0001    //  driver message buffer is flooded
```

Data number (start)

DN_AUDIO_SETSTATUS

Argument

`buf`

Pointer to the UW-type variable where the new internal driver status is stored

`size`

Must be `sizeof(UW)`.

2.5.5 Output data format designation

Designates the data format for blocks to be written.
(Note: The supported format is dependent on the hardware to be implemented.)

Data number `(start)`

DN_AUDIO_SETOUTPUTFMT

Argument

`buf`

Pointer to the following `AudioDriverDataFormat` structure

```
struct {
  W     nSize;
  W     nFormatTag;
  W     nFS;
  W     nChannels;
  W     nInterleaveSample;
} AudioDriverDataFormat;
```

`nSize`

Structure size of AudioDriverDataFormat [byte]

`nFormatTag`

Designates the sample format type (if rawPCM, the number of bits, the byte order, and whether it is signed or unsigned).
Designates specific values for each of the various combinations.

Examples:

FMT_PCM_S16_LE	rawPCM signed 16bit LittleEndian
FMT_PCM_U8	rawPCM unsigned 8bit (offset binary)

nFS
Designates the sampling rate [Hz].

nChannels
Designates the number of channels (>=1) in the data stream.

nInterleaveSample
If nChannels >1 for the data stream, designates after how many samples the channels are changed.
Examples:
When nChannels = 2

	Time---------->
nInterleaveSample=1	L R L R L R L R L R...
nInterleaveSample=4	L L L L R R R R L L...

size
Must be sizeof(AudioDriverDataFormat).

2.5.6 Input data format designation

Designates the data format for blocks to be read.
Designated the same way as for the output data format.
(Note: The supported format is dependent on the hardware to be implemented.)

Data number (start)
DN_AUDIO_SETINPUTFMT

Argument
buf
Pointer to the AudioDriverDataFormat structure
For a description, see the section on output data format.
size
Must be sizeof(AudioDriverDataFormat).

2.5.7 Output drive control (optional)

Controls the operating status of audio output hardware (including DAC hardware).
The default operating status is always run.
The function is intended for situations in which the timing of initial data writing and the moment sound is heard must be separately controlled. This is the case for hardware with a significant delay (latency) in the period after specific data is written using DN_AUDIO_PLAYAUDIO until the time audio starts to be heard.

Data number (start)
DN_AUDIO_SETOUTPUTSTATE

Argument
buf

Pointer to the UW-type
The UW-type indicated by buf prescribes the following operating parameters.

bit 31	0 = stop 1 = run
bit 30 to 16	reserved (must be 0)
bit 15 to 0	implementation-dependent

size
Must be sizeof(UW).

2.5.8 Input drive control (optional)

Controls the operating status of audio input hardware (including ADC hardware).
The default operating status is always run.
The function is intended for situations in which the timing of indication of initial data reading and recording must be separately controlled. This is the case for hardware with a significant delay (latency) in the period after specific data is read using DN_AUDIO_RECAUDIO until the time audio starts to be recorded.

Data number (start)
DN_AUDIO_SETINPUTSTATE

Argument
buf
Pointer to the UW-type
The UW-type indicated by buf prescribes the following operating parameters.

bit 31	0 = stop 1 = run
bit 30 to 16	reserved (must be 0)
bit 15 to 0	implementation-dependent

size
Must be sizeof(UW).

2.5.9 Mixer functions (optional)

This section defines mixer functions for write operations.
- In this specification, "mixer" refers to volume adjustment functions and analog mixing functions. (See Section 2.7.)
- Mixer functions can only be controlled with subunit 0. Attempting to use them with subunit 1 or later returns an error (E_OBJ).
- Although software-based volume control can be implemented for platforms that do not support analog mixing hardware, latency may affect the audio output.
- All mixer states immediately after open are undefined.

2.5.9.1 Set output volume

Controls the output volume of the designated line.

Data number `(start)`

DN_AUDIO_MIXERSETOUTPUTVOL

Argument

`buf`

Pointer to the following `MixerLineVolume` structure

```
struct {
  UB   lineId;// line ID
  UB   time;  // the time spent changing the volume [msec]
  H    vol[0];// the volume value
} MixerLineVolume;
```

The following kinds of ID are designated for the line.

`MIXER_LINEID_MASTEROUT`	The output master volume
`MIXER_LINEID_PCMOUT`	PCM volume
`MIXER_LINEID_MICIN`	Microphone volume

Each channel's volume is designated after vol[0].

Examples:

For a monaural line

`vol[0]` = volume

For a two-channel stereo line

`vol[0]` = L channel volume

`vol[1]` = R channel volume

The volume value is designated as a logarithm in increments of 1/256 dB. The volume value for the nth channel is `(vol[n]÷256) dB.`

The volume value is clipped to be in the effective range for that line.

In time, the time spent until the designated volume value is reached is designated. To reach the designated volume value immediately, 0 is designated for time.

When time>0, the driver gradually changes the volume from the current volume, taking time [msec] to reach the volume value designated after vol[0]. This function is intended to suppress quick noise, and implementation is optional. If quick noise suppression functions are implemented in the hardware, time is ignored.

`size`

The effective byte size from buf

2.5.9.2 Set input volume

Controls the input volume (recording level) of the designated line.

Data number `(start)`

DN_AUDIO_MIXERSETINPUTVOL

Argument

`buf`

Pointer to the `MixerLineVolume` structure
For a description, see the section, `"set output volume."`

`size`

The effective byte size from buf.

2.5.9.3 Mute line

Mutes or unmutes the output of the designated line.
The volume value is retained even if muted. Volume adjustment is possible, but there is no sound for this line until unmute is used.

Data number `(start)`

DN_AUDIO_MIXERMUTELINE

Argument

`buf`

Pointer to the `UW`-type
The UW-type indicated by `buf` prescribes the following operating parameters.

`bit 31`	0 = unmute 1 = mute
`bit 30 to 16`	reserved (must be 0)
`bit 15 to 8`	time spent for mute [msec]
`bit 7 to 0`	line ID

For bit 15 to 8, the time spent until muting [msec] is designated as an unsigned 8-bit integer. For instantaneous mute, designate 0.
When (bit 15 to 8)>0, the driver gradually lowers the volume from the current level until it is silent. This function is intended to suppress quick noise, and implementation is optional. If quick noise suppression functions are implemented in the hardware, bit 15 to 18 is ignored.

`size`

Must be `sizeof(UW)`.

2.5.9.4 Select recording source

Sets a recording source for the designated line.

Data number `(start)`

DN_AUDIO_MIXERSELECTRECSRC

Argument

`buf`

Pointer to the following `MixerLineRecSrc` structure

```
struct {
  W     nLines;
  UB    lineId[0];
} MixerLineRecSrc;
```

`nLines`

Number of lines designated for the recording source

`lineId[0] to [nLines-1]`

Line ID for the lines to be used with the recording source

`size`

The effective byte size from buf

2.6 ID tk_rea_dev(ID dd, INT start, VP buf, INT size, TMO tmout) ER tk_srea_dev(ID dd, INT start, VP buf, INT size, INT *asize)

2.6.1 Audio recording

Designates the start address of the buffer and number of buffer blocks for recording.
If requests are queued through asynchronous processing, the driver must enter processing for the next request continuously after the current request is complete without pause.
A buffer must not be released once it has issued a read request until that request is complete.
If the driver message buffer (see Section 2.5.2) is registered by DN_AUDIO_REGISTERMSGBUF, the driver issues a notification message immediately before and after recording to the buffer.

Data number `(start)`
DN_AUDIO_RECAUDIO (=0)

Argument

`buf`
The start address of the buffer where recording data is stored.
Depending on the hardware platform, the following limitations may occur.
(Examples of limitations)
Alignment required for each memory page
Must not be in task-specific space

`size`
The number of buffer blocks.

2.6.2 Get the supported data format

Lists the data formats supported by the device unit.
Descriptions of data formats are defined separately.
If the available array is not large enough, E_PAR is returned.

Data number `(start)`
DN_AUDIO_GETAVAILABLEFMTS

Argument

`buf`
Pointer to the `B`-type array
The data format description is stored in the array indicated here as a half-width alphanumeric (ASCII) string.

`size`
The size of the array indicated by buf [bytes]

2.6.3 Get the internal driver status

Gets the driver-specific internal status.

The internal driver status is copied to the UW-type variable indicated by buf.
Each bit of the internal driver status is significant. For telling what the status is, check each bit like as (status & AUDIO_STATUS_MBFFLOW).

```
#define AUDIO_STATUS_MBFFLOW  0x0001    // driver message buffer is flooded
```

Data number `(start)`
DN_AUDIO_GETSTATUS

Argument
`buf`
Pointer to the `UW`-type variable
`size`
Must be `sizeof(UW)`.

2.6.4 Get the current write address

Gets the current write address of the buffer during recording.
If recording is not in progress, E_OBJ is returned.

Data number `(start)`
DN_AUDIO_GETRECORDINGPOS

Argument
`buf`
Pointer to the `VP`-type variable
The current write address of the buffer is stored in the variable indicated here.
`size`
Must be `sizeof(VP)`.

2.6.5 Get the current read address

Gets the current read address of the buffer during playback.
If playback is not in progress, E_OBJ is returned.

Data number `(start)`
DN_AUDIO_GETPLAYINGPOS

Argument
`buf`
Pointer to the `VP`-type variable
The current read address of the buffer is stored in the variable indicated here.
`size`
Must be `sizeof(VP)`.

2.6.6 Mixer functions (optional)

This section defines analog mixer functions for read operations.

- In this specification, "mixer" refers to volume adjustment functions and analog mixing functions. (See Section 2.7.)
- Mixer functions can only be controlled with subunit 0. Attempting to use them with subunit 1 or later returns an error (E_OBJ).

2.6.6.1 Enumerate lines

Lists mixer elements (supported line numbers, minimum and maximum volume, and name) as well as the number of channels supported by each line.
The descriptors of each line are `lineDesc[0]` to `[nLines-1]`.

Data number `(start)`

DN_AUDIO_MIXERENUMLINES

Argument

`buf`

Pointer to the following `MixerAllLinesDesc` structure

```
struct {
 UB lineId;      // line ID
 UB nChannels;   // number of channels
 H  volMax;      // maximum effective volume [1/256 dB]
 H  volMin;      // minimum effective volume [1/256 dB]
 B  LineName[32];// line name (half-width alphanumeric characters)
} MixerLineDesc;

struct {
 W  nLines;      // number of lines assigned to the mixer
 struct MixerLineDesc LineDesc[0];
} MixerAllLinesDesc;
```

2.7 Mixer hardware example (optional)

A codec is assumed for this specification that supports multiple analog input mixing functions. Figure 4 shows an example of this. The mixer is depicted by the area bounded by dotted lines in the figure.
As a specific example, Figure 5 shows the portion representing a mixer in a configuration supporting one stereo output system and one monaural input system.
These are merely provided as examples of mixers supported by a driver of this specification. They are not intended to limit the hardware corresponding to the driver.

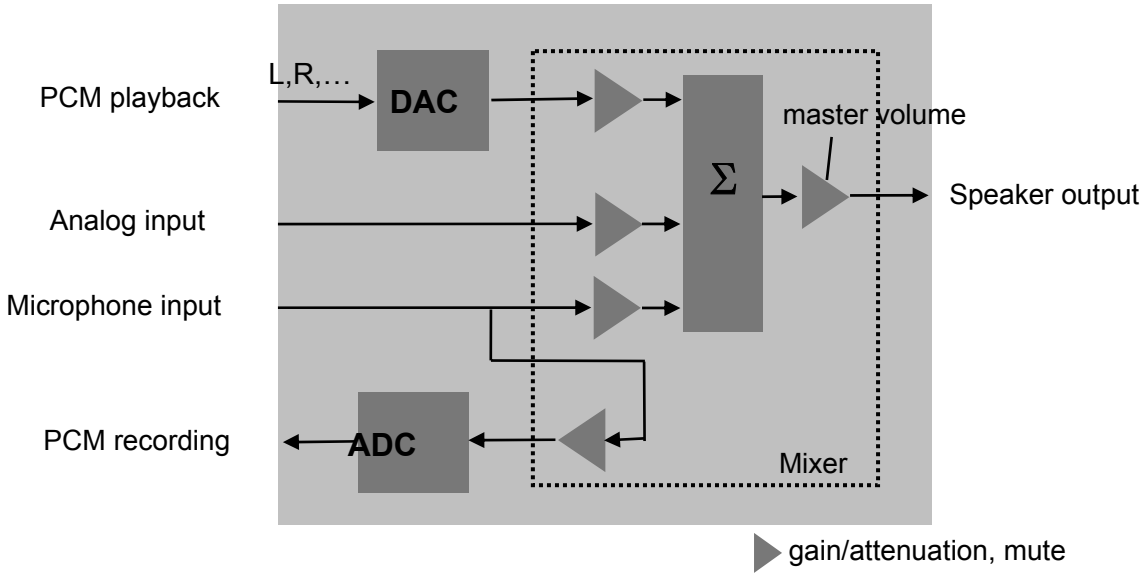

Figure 4. Example of a codec assumed for this specification

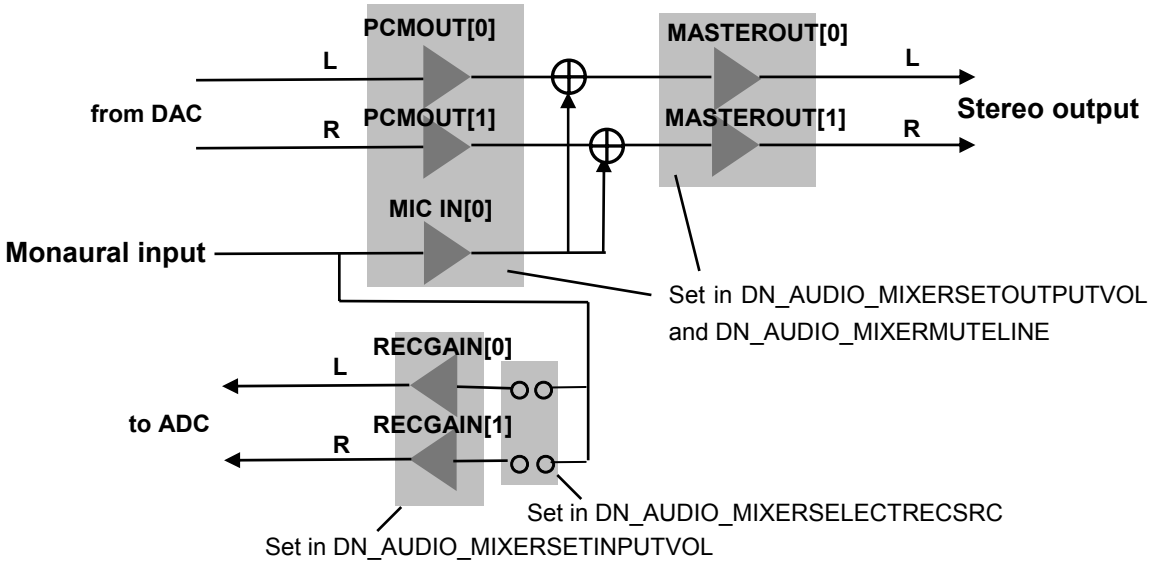

Figure 5. Example of a mixer with one stereo output system and one monaural input system

2.8 Device control code example (concept)

2.8.1 Without request queuing

```
          •
          •
          •
// opens the device in write mode
dd = tk_opn_dev("audioa0", TD_WRITE);
// designates PCM format
tk_swri_dev(dd, DN_AUDIO_SETOUTPUTFMT, &pcmfmt, sizeof(pcmfmt), &asize);
// initializes the mixer volume
// master volume
tk_swri_dev(dd,DN_AUDIO_MIXERSETOUTPUTVOL,&mastv,sizeof(mastv),&asize);
tk_swri_dev(dd,DN_AUDIO_MIXERMUTELINE,&mastmon,sizeof(mastmon),&asize);
// pcmout volume
tk_swri_dev(dd, DN_AUDIO_MIXERSETOUTPUTVOL, &pcmv, sizeof(pcmv), &asize);
tk_swri_dev(dd, DN_AUDIO_MIXERMUTELINE, &pcmmon, sizeof(pcmmon), &asize);
// audio playback
// playback from buf of length [block]
tk_swri_dev(dd, 0, buf, length, &asize);
// mutes the master volume (stopping noise)
tk_swri_dev(dd,DN_AUDIO_MIXERMUTELINE,&mastmoff,sizeof(mastmoff),&asize);
// closes the device
tk_cls_dev(dd, 0);
          •
          •
          •
```

2.8.2 With request queuing

```
(Up to this point, same as without queuing)
// audio playback
nBuf = 0;
id[nBuf] = tk_wri_dev(dd, 0, buf_A, length, TMO_FEVR); // id is the ID type,
an array 2 long
// repeats until there is a termination request
while (!bEnd){
   // handles the sound buffer
   audioproc(nBuf?buf_A:buf_B);
   // issues the next request (with queuing)
   id[nBuf^1] = tk_wri_dev(dd, 0, nBuf?buf_A:buf_B, length, TMO_FEVR);
   // waits for termination of current request
   tk_wai_dev(dd, id[nBuf], &asize, &er, TMO_FEVR);
   // switches the buffer
   nBuf ^= 1;
}
tk_wai_dev(dd, id[nBuf], &asize, &er, TMO_FEVR);
          •
          •
```